HAW HAMBURG

BACHELOR THESIS
Aldo Bega

Investigating the Role of AI in Developing a Cross-Platform Book Recommendation Application

FACULTY OF ENGINEERING AND COMPUTER SCIENCE
Department of Information and Electrical Engineering

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Aldo Bega

Investigating the Role of AI in Developing a Cross-Platform Book Recommendation Application

Bachelor Thesis based on the examination and study regulations
for the Bachelor of Engineering degree programme
Bachelor of Science Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner: Prof. Dr. Martin Lapke
Second examiner: Prof. Dr. Marc Hensel

Day of delivery: 6. March 2025

Aldo Bega

Title of Thesis

Investigating the Role of AI in Developing a Cross-Platform Book Recommendation Application

Keywords

Artificial Intelligence, Book Recommendation Systems, Java Spring Boot, Cross-Platform

Abstract

This research is dedicated to observing the role of AI in the development of a cross-platform book recommendation application, stressing the use of innovative technologies to enhance the user interface. A combination of Java and Spring Boot as server-side technologies is implemented to develop a reliable yet scalable infrastructure. Simultaneously, Flutter is used to make the system run across different devices and interfaces. MySQL is the relational database used to manage the data of the user, lists, books, and interactions effectively.

The research paper analyzes the technical hurdles met during the process of merging AI functions with traditional software components that include the guarantee of real-time data processing and also the protection of user data. Through a user-centered approach, the paper assesses how user engagement is fueled by AI and a level of satisfaction in literary applications is achieved. It further investigates the capacity of cross-platform development frameworks like Flutter for eliminating the development overhead while retaining high-quality standards.

Aldo Bega

Thema der Arbeit

Untersuchung der Rolle der KI bei der Entwicklung einer plattformübergreifenden Anwendung zur Buchempfehlung

Stichworte

Künstliche Intelligenz, Buch-Empfehlungssysteme, Java Spring Boot, Plattformübergreifend

Kurzzusammenfassung

Diese Forschung widmet sich der Beobachtung der Rolle der KI bei der Entwicklung einer plattformübergreifenden Buchempfehlungsanwendung und betont den Einsatz innovativer Technologien zur Verbesserung der Benutzeroberfläche. Eine Kombination aus Java und Spring Boot als serverseitige Technologien wird implementiert, um eine zuverlässige und dennoch skalierbare Infrastruktur zu entwickeln. Gleichzeitig wird Flutter verwendet, um das System auf verschiedenen Geräten und Schnittstellen laufen zu lassen. MySQL ist die relationale Datenbank, mit der die Daten des Benutzers, Listen, Bücher und Interaktionen effektiv verwaltet werden.
Die Forschungsarbeit analysiert die technischen Hürden, die beim Zusammenführen von KI-Funktionen mit herkömmlichen Softwarekomponenten auftreten, einschließlich der Garantie der Echtzeit-Datenverarbeitung und des Schutzes von Benutzerdaten. Durch einen benutzerzentrierten Ansatz bewertet die Arbeit, wie das Engagement der Benutzer durch KI gefördert und ein gewisses Maß an Zufriedenheit bei literarischen Anwendungen erreicht wird. Sie untersucht außerdem die Fähigkeit plattformübergreifender Entwicklungsframeworks wie Flutter, den Entwicklungsaufwand zu eliminieren und gleichzeitig hohe Qualitätsstandards beizubehalten.

Contents

List of Figures

1 Introduction

1.1 Background

The last few years have seen massive shifts in literature consumption owing to new digital technologies such as e-books, online libraries, and book recommendations. The universe of e-books, reader reviews, and accessing digital libraries has changed the reading experience. Now, readers are in need of platforms where they will be served diverse content and enabling them to express their preferences. The transformation will be incomplete without mobile apps, which are a must in the whole process.

D-espite the numerous digital platforms available, the gap remains for a far-reaching mobile application that can bring the main features of books, such as recommendations, reviews, summaries, and reading lists, together into a user-friendly interface. Artificial intelligence (AI) has played a big part in machine learning and natural language processing, creating chances to close this gap by analyzing user preferences and reading behaviors. With the help of AI, applications can automatically suggest a list of personalized books based on user preferences. Therefore, they contribute to the reading experience. [14]

The project's goal is to fill this gap by creating an AI-enhanced cross-platform application that brings together all these features and gives book lovers a platform for unity. The app is intended to provide recommendations based on the user's preferences, allow an in-depth analysis of the content in the user's engagement, and provide a more personalized literature interaction.

1.2 Motivation

In an era where more information is coming out every day, finding the perfect book out of many options that suits an individual's taste is no walk in the park. The amount of books to choose from is staggering, which results in readers most of the time not being able to know if they have left out the ones the most interesting to them because there are no personalized suggestions. Moreover, it is now a trending phenomenon to want to discover the essence of a book in a few but adequate words so that the reader can decide whether to proceed with reading it or not. The main aim of the app is to ease the issues that are associated with the book excerpt through solving them with a single neat tool.

1.3 Objectives

This research's central goal is to develop and design a cross-platform mobile application using AI, which is designed to facilitate the interaction of readers with the world of books. To enable AI-driven functions, the app aims to be a comprehensive platform through which users can find books more easily. More specifically, this research is directed at:

1. **Establish an AI-based book recommendation system** whichm will apply AI algorithms to understand user preferences and favorites with the help of metadata, as well as provide recommendations that suit individuals' personal tastes.
2. **Design and implement a strong review system** allowing the users to share their impressions of the books they read. On the other hand, users can also seek out the thoughts and recommendations of other readers, thus making the platform a user-driven initiative in sharing literary viewpoints and building a strong network of book lovers.
3. **Provide an effective book summary** tool by bringing up the summaries from the database, which allows users to get quick and concise knowledge about a book's theme, idea, and content. This function should not only facilitate users in making a decision about what to read but also help people with less time to get informed about books that they are interested in.
4. **Create a connection between technology and writing** by clearly showing how AI's capabilities can be used to make books more accessible and engaging to a larger group of people. The application will underscore the fact that technology not only promotes book discovery but also allows people to connect on a deeper level with literary works through innovative and engaging ways of exploring words and phrases in the literary world.

2 Theory

2.1 Overview of Key Concepts

Book Recommendation Systems

Book recommendation systems are a type of recommendation system that is designed especially to offer users personalized book suggestions. These systems analyze various data points which also include user preferences and reading history along with demographic information to give out customized recommendations. The unique goal of these systems is to increase user satisfaction and engagement by suggesting books that fit their unique taste.

The most employed methods are collaborative filtering and content-based filtering, with the first one being a system that detects the similarities of user's choices and suggests books that are enjoyed by others with similar tastes. On the other hand, the second method is composed of attributes like the books' genre, author, and keywords to find books that match the taste of the users. Many advanced systems now adopt hybrid approaches, combining collaborative and content-based methods to improve accuracy and diversity in recommendations. [14]

The use of the machine learning and artificial intelligence in the book recommendation systems has really changed the whole game as the systems are more and more adaptable to the user behavior over time. The new technological advances not only allow the systems to handle dynamic data but also make them more accurate in predicting user preferences while they can offer real-time recommendations. Such systems, therefore, are the backbone of digital libraries, online bookstores, and book-centric mobile applications. [1]

Artificial Intelligence in Recommendation Systems

Artificial intelligence (AI) is the major force behind the upgrade of modern recommendation systems. These systems can process large amounts of data and provide highly accurate and personalized proposals by using machine learning algorithms. Collaborative filtering, content-based filtering, and hybrid models are the basic methods to make these systems [16]

Recent achievements in artificial intelligence, including natural language processing (NLP) and deep learning improved the abilities of recommendation systems. These improvements enable data systems to contextualize user data, understand the implicit behavior, and provide recommendations that are not just personalized but also contextually relevant. [16]

In this project, AI powers book recommendations by analyzing user inputs, such as favorite book lists, and generating insightful suggestions. Additionally, AI algorithms enable the dynamic adaptation of the system to evolving user preferences, ensuring continuous improvement and user satisfaction. [8]

Cross-Platform Mobile Application Development

Cross-platform mobile development has revolutionized how applications are built by enabling developers to write a single codebase that works across multiple platforms. This approach significantly reduces development time, costs, and the complexity of maintaining separate codebases for Android, iOS, and other platforms. [17]

Google's Flutter, a UI toolkit, is one of the most advanced and influential in the world of cross-platform development at the moment. Its widget-based architecture results in an easy-to-use system that generates uniform, responsive, and user-friendly application interfaces. Furthermore, another point about the technology is that Flutter also runs applications natively, so the applications perform very well on the devices they run on. The Dart programming language, which is the technology of Flutter, is a new and efficient one, and it is specifically designed for developing applications that can be expanded rapidly. [4]

For this project, Flutter ensures that the mobile application is accessible to a broad audience while maintaining a high-quality user experience. Its flexibility and rich ecosystem allow developers to integrate AI-powered features effectively, ensuring that the application is functional and visually appealing.

OpenAI API

The OpenAI API is a versatile tool for implementing advanced AI functionalities in applications. Powered by GPT models, the API excels in natural language understanding and generation, making it suitable for a wide range of applications, including recommendation systems, chatbots, and content creation. [7]

The OpenAI API is a key feature of this project, serving as the main mechanism for book recommendations. It analyzes user preferences and data to make intelligent suggestions. The GPT models' ability to interpret challenging inputs and provide context-based outputs makes them vital.

The main research point of this project is the OpenAI API call, which is the main facilitator in the development of book recommendations. Through the use of this API, which uses GPT models, the user can obtain a highly interactive interface where it is possible to receive and act on the outputs obtained.

The advent of OpenAI API to the system not only brings in the feature of adaptability but also enhances its robustness by continuously updating its suggestions with newly added user data. Through this API, the application provides dynamic and engaging recommendations, different from traditional rule-based systems.

Backend Development with Java and Spring Boot

The critically acclaimed Spring Boot has numerous assets that emphasize the model, particularly its modular architecture, ease of configuration, and extensive library ecosystem, which is why it is a widely used framework among developers. Its primary aim is to make the development of strong and scalable applications simpler. That is why Spring Boot is very effective in building and creating RESTful APIs. These APIs are widely used in the mobile and web development industry. [15]

The platform independence of Java, along with Spring Boot's functionalities, empower developers to establish backend solutions that are both sustainable and scalable. Through microservices architecture, the framework can allow applications to deal with the most complicated requirements and scale properly as they develop. Besides that, Spring Boot's compatibility with both relational and non-relational databases leads to data flexibility, which enables developers to come up with efficient data storage systems that are also safe. [15]

Spring Boot ensures seamless communication between the mobile frontend and backend systems for this project. It provides the infrastructure needed to process user requests, retrieve data, and integrate AI functionalities, ensuring a smooth and responsive user experience.

2.2 Existing Solutions and Relevant Literature

Existing Book Recommendation Systems

Goodreads, Amazon Kindle, Scribd, and other platforms have book recommendation features designed to help users discover new titles. These models are essentially structured traditional recommendation algorithms with inherent limitations, making them only partly useful.

The majority of recorded activities on such platforms apply common collaborative filtering methods. This involves ”user behavior dataset filtering,” where the system predicts a user’s taste by comparing ratings and habits of different users. However, this method focuses on popularity-based factors and often overlooks niche or diverse books that could be of interest. These platforms fail to consider the depth of a person’s likes and dislikes, such as mood, reading goals, or interest in an unfamiliar genre.

Moreover, these systems lack features like smart prediction of users’ wishes and moods reflected in their reading. They also struggle with the ”cold start” problem when new users or book data are too limited, leading to generalized recommendations. Beyond that, they typically do not adapt to newly forming preferences.

The lack of deep personalization in recommendation systems is a cause for concern. This highlights the need for more intelligent AI-driven systems that use machine learning, natural language processing, and contextual analysis to improve recommendations, making them more accurate, diverse, and responsive to changing user preferences.

Relevant Academic Work

Recent studies on recommendation systems and AI emphasize hybrid methods for increasing accuracy and relevance. A combination of collaborative filtering and content-based techniques, like analyzing book genres, themes, and writing styles, results in a more comprehensive system. These hybrid models enhance efficiency by reducing the weaknesses of each methodology. Collaborative filtering identifies user patterns, while content-based strategies focus on item properties. Together, they create a cohesive personalized system. [14]

Sophisticated AI algorithms, such as neural networks and natural language processing, are increasingly used to adapt to users’ evolving preferences and provide deeper contextual understanding. AI-fueled hybrid systems help address the cold start problem and biases of traditional methods. [14]

Additionally, literature suggests a platform-neutral approach, ensuring users can switch between devices seamlessly. Cross-platform applications are essential for maintaining user satisfaction and engagement by providing a consistent experience across all devices.

Hybrid recommendation systems, combined with cross-platform functionality, offer highly personalized recommendations in the easiest way. AI enables apps to evolve dynamically with users' changing consumption patterns, making them more effective and adaptable.

2.3 Explanation of Technologies

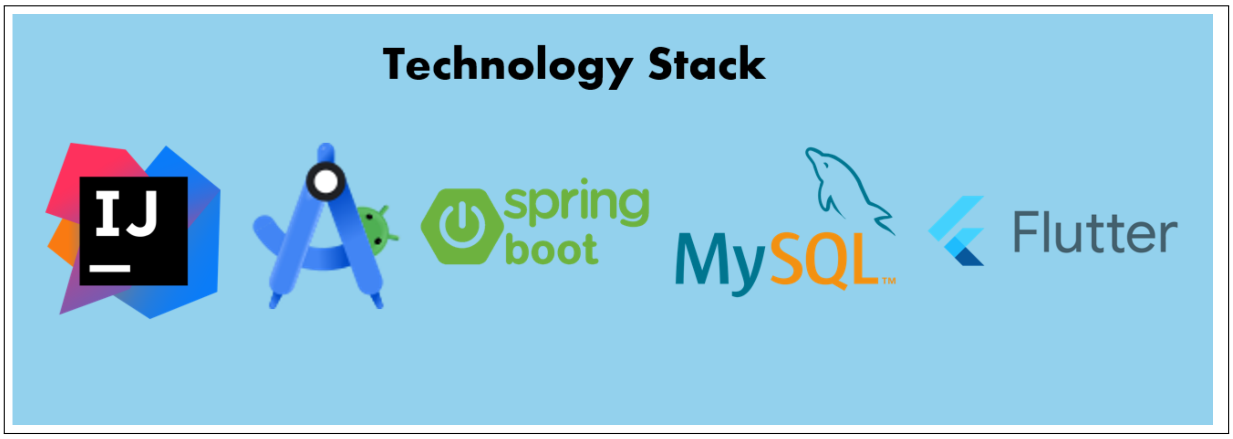


Figure 2.1: Technology Stack

2.3.1 OpenAI

By deploying state-of-the-art natural language processing (NLP) features, OpenAI's API facilitates applications to inspect, process, and compose human-like text with precision. Utilizing cutting-edge NLP techniques, it grasps language concepts, and context—essential for developing intelligent recommendation systems. Integrating this into a book recommendation platform allows for more complex suggestions based on user feedback, loved books, genres, and topics.

What sets the OpenAI API apart is its ability to process complex descriptions and use relevant knowledge to suggest books most likely the user is interested in. Users input favorite titles, and the AI analyzes rankings to find patterns. This enables recommendations based on style, tone, and content, representing the next stage in personalization.

2.3.2 MySQL

MySQL is a highly reliable relational database management system that ensures data consistency, integrity, and scalability across applications. Being open-source, it has gained widespread adoption due to its simple interface and strong SQL-based query support, making it ideal for managing structured data. [12]

MySQL excels in handling complex relationships like user profiles, book details, ratings, and reviews in recommendation systems. Its ability to maintain normalized tables minimizes redundancy while ensuring logical entity relationships. Fully ACID-compliant, it guarantees secure data storage, even during concurrent transactions or system failures. [12]

Designed for large-scale applications, MySQL supports indexing and optimized queries, reducing processing time for quick data retrieval. Its scalability, through vertical and horizontal scaling, replication, and clustering, ensures responsiveness under heavy workloads. Additionally, MySQL integrates with tools like Hibernate and JPA, simplifying backend development. Its speed, reliability, and flexibility make it a crucial element in delivering efficient and secure user experiences.

2.3.3 Flutter

Flutter, backed by Google, is a fast and efficient cross-platform framework for mobile applications. Using Dart, it enables natively compiled code for both iOS and Android, ensuring high-speed performance while maintaining cross-platform benefits. One of its standout features is a rich set of customizable widgets that create attractive, interactive user interfaces, adaptable to any screen for a seamless experience. [6]

A major advantage of Flutter is its single codebase, reducing development complexity and ensuring faster cycles with easy maintenance. Its ability to support both platforms with one core technology makes it a favorite among developers. [6]

Flutter's growing ecosystem includes libraries, tools, and plugins for animations, API calls, and real-time data processing. With support from Google and the open-source community, it remains a top choice for developers, allowing them to build high-quality, feature-rich mobile applications efficiently. [6]

2.3.4 Java and Spring Boot

Java is a versatile, object-oriented programming language widely recognized for its platform independence, reliability, and scalability. Its "Write Once, Run Anywhere" (WORA) capability, enabled by the Java Virtual Machine (JVM), allows applications to run seamlessly across multiple operating systems without modification. Java's strong memory management, multithreading capabilities, and extensive library support make it ideal for developing secure and high-performance applications, from web services to large-scale distributed systems. Additionally, its backward compatibility and strong community support ensure continuous improvements and long-term stability. [3]

Spring Boot enhances Java's capabilities by simplifying backend development, offering auto-configuration and built-in tools for database access, security, and messaging. It reduces boilerplate code, allowing developers to focus on business logic while ensuring rapid development. One of its standout features is the ability to create RESTful APIs efficiently, facilitating seamless communication between the backend and various clients, including mobile apps, web interfaces, and third-party services. [15]

With built-in support for microservices, robust security features, and a wide range of extensions, Spring Boot is an optimal choice for modern backend systems. Its adaptability and efficiency empower developers to build scalable, maintainable, and high-performing applications that meet evolving business demands. [15]

2.3.5 REST API

A RESTful API (Representational State Transfer API) is an architectural style that is designed to allow communication between the client and the server by using the standard protocols of the web, primarily HTTP. Nowadays RESTful APIs are the most commonly developed APIs used, especially in the field of software development, and this happens due to their scalability, stateless nature, and a special ability to integrate perfectly between different platforms and services. [11]

RESTful APIs operate on a client-server model, where the client (e.g., a mobile app or web application) sends HTTP requests to access or modify resources on the server. These requests follow standard HTTP methods:

- GET – Retrieves data from the server.
- POST – Submits new data to the server.
- PUT – Updates an existing resource.
- DELETE – Removes a resource from the server

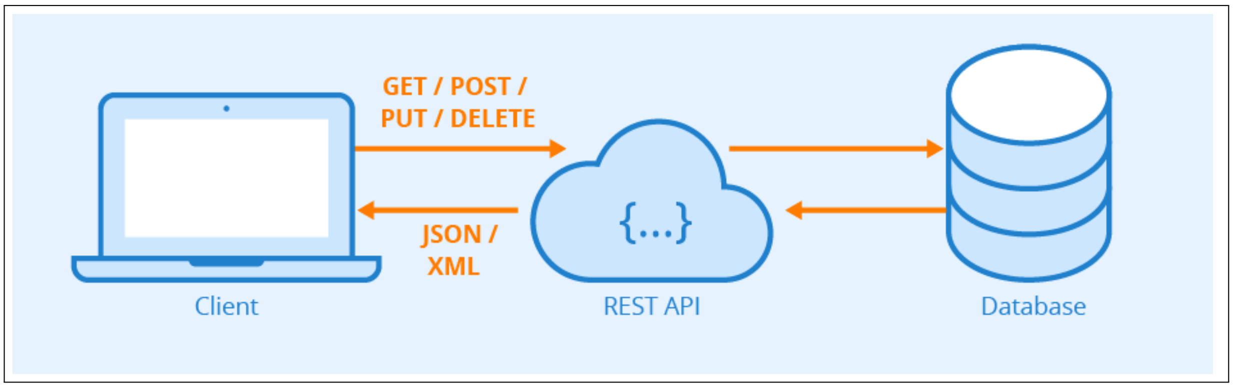


Figure 2.2: RESTful API Diagram [2]

RESTful APIs are well known for their statelessness, which is their main feature, as they do not keep any new information about the context of the request from the client. That means anyone can implement REST APIs and create very scalable and efficient systems as the server does not have to remember all the conditions that the session is in. [11]

The key benefits of RESTful APIs are:

- Scalability – While they don't keep a state, REST APIs still can handle a lot of client requests and do not maintain session data on the server.
- Interoperability – REST APIs utilize the standard data formats known as JSON and XML, and thus, they can easily interoperate with multiple programming languages and platforms.
- Flexibility – REST APIs enable clients to ask for only the very essential data that is why it works faster and reduces network stress.
- Security – REST APIs can affirm the user through access control strategies like OAuth, JWT, API key, etc., to cut off the unauthorized person and protect the sensitive data.

3 Concept and Requirements

3.1 Concept Overview

The aim of the planned app is to provide a platform that is truly comprehensive for book enthusiasts and to offer multiple possibilities for their experience, such as finding a new book, being provided with a personal recommendation, or engaging with other users, even in the case of the review writing process. The app powered by AI will recommend the most suitable books to users by analyzing the books that the individual reader has in its favorites list, offering the best books only for the specific user, not everyone. It will not only be a user-friendly platform but it will also be constructed with high attention to accessibility, which is necessary to make the experience smooth to the end-user with different devices and platforms. The app is set to make the whole process of book discovery even more efficient, attractive, and deeply personalized by integrating different characteristics. The goal is to create the means through which readers discover new books more efficiently and interactively.

3.2 User Stories

User Authentication and Account Creation:

Story 1: As a new user, I want to sign up using my email address, so that I can create an account and access personalized features of the app.

Story 2: As a returning user, I want to log in to my account with my credentials, so that I can resume my progress and access my saved data.

Story 3: As a user who forgot my password, I want an option to securely reset my password through my registered email, so that I can regain access to my account.

Book Discovery and Recommendations:

Story 1: As a new user, I want to see the top rated books as soon as I open the application.

Story 2: As a reader, I want the app to recommend books based on my preferences, so I can discover books that match my taste.

Story 3: As a user looking for variety, I want the app to suggest trending or popular books, so I can find what others are reading and discussing.

Book Reviews and Summaries:

Story 1: As a reader, I want to write and publish reviews for books I've read, so I can share my opinions with other users.

Story 2: As a user looking for quality reviews, I want to read the most helpful or relevant reviews, so I can get an accurate sense of the book before reading.

Story 3: As a user, I want access to concise summaries of books, so I can understand their key points.

Search and Filter:

Story 1: As a reader with a specific book in mind, I want to search by title or author, so I can find information about the exact book I am looking for.

Story 2: As a genre-focused reader, I want to see all the genres available, and also be able to search for a specific genre.

Story 3: As a reader who likes variety, I want the app to suggest similar books when I search for or view a book, so I can discover related options.

3.3 Use Cases

The below use case diagram denotes the core functionalities of the book recommendation application and shows the point-to-point interactions among the user and the system. It involves the main things such as user authentication, book finding, AI-driven recommendations, review management, and personal library organization. The diagram also demonstrates the role of the AI engine in the selective process of personalized book suggestions.

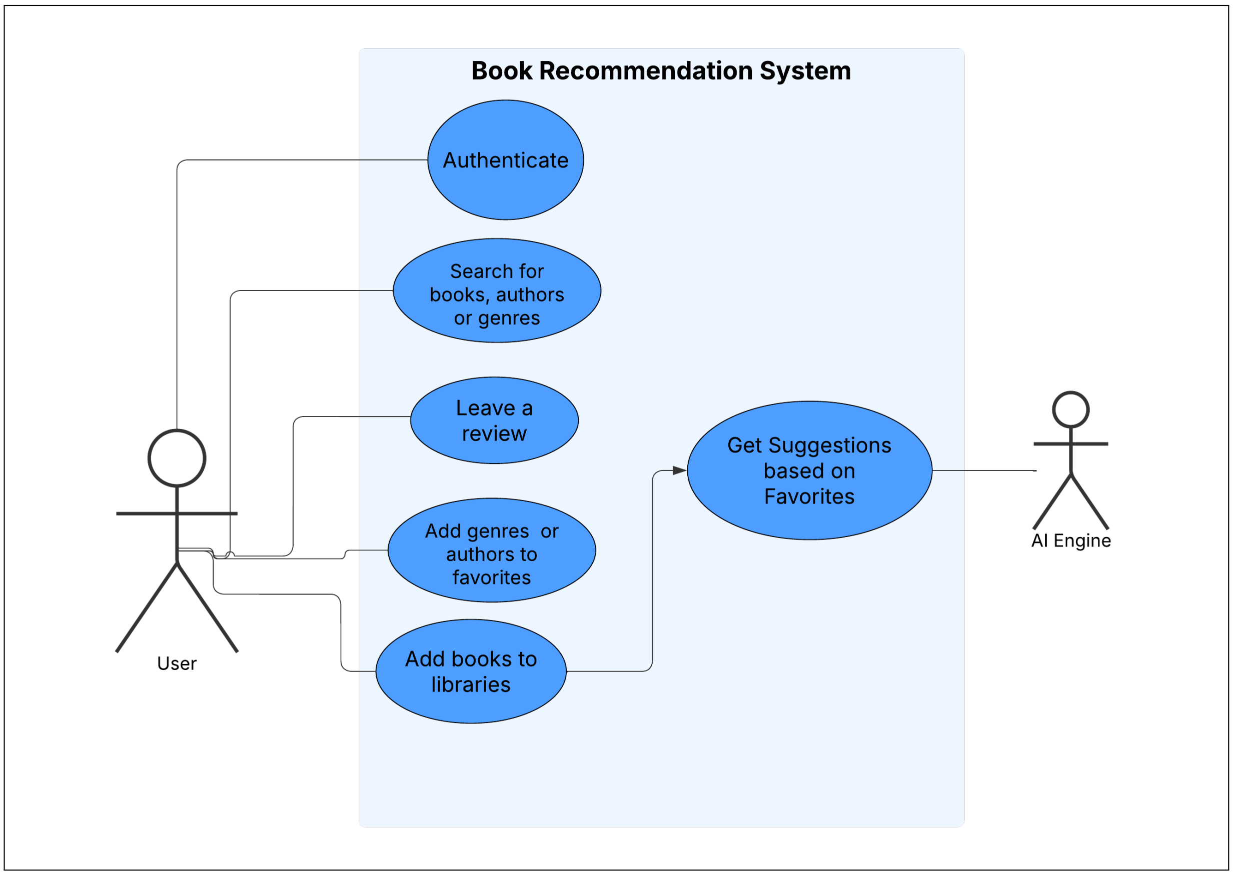

Book Recommendation System
Authenticate
Search for books, authors or genres
Leave a review
Add genres or authors to favorites
Add books to libraries
Get Suggestions based on Favorites
User
AI Engine

Figure 3.1: Use Case Diagram

3.4 Functional Requirements

A book recommendation app's functional requirements get to the point of what features and interactions are to be implemented and where they must be included to complete the user's requirements and retrieve information quickly. These requirements concentrate on the main functions that the application has to deal with, namely a user login, personal suggestions, a review section, and searching mechanisms. Each requirement is created to make the app more easily operatable, secure the necessary benefits to the clients, and give them feedback that is specifically driven by artificial intelligence on content and reading levels in a particular area. In combination, these functional features make up the groundwork for the app's functionalities, thus making the app very suitable for the discovery of new interesting books.

User Profile and Preferences

Users must be able to create profiles where they can set and update their reading preferences. The user should be able to easily change their preferences, in order to retrieve different suggestions.

Personalized Recommendation System

The app must provide book recommendations based on user preferences. It should suggest personalized recommendations as specific as possible, meaning that for each change in the user's favorite book list, the app should make different suggestions.

Book Review System

Users must be able to write and publish reviews for books within the app. The app should enable users to read reviews written by others see the date and rating of the these reviews.

Book Summarization Feature

The app must offer summaries of books that capture the main ideas and themes. These summaries will be directly fetched from the database.

Search

The app must include a search function that allows users to find books by title, author, or genre.

Reading List Management

The app must provide a feature where users can create and manage lists of books they have read, are currently reading, or want to read in the future.

3.5 Non-Functional Requirements

These non-functional requirements associated with the context of this book recommendation application are the essential qualities and performance criteria that ensure the reliability and responsiveness of the user-friendly experience. As opposed to functional requirements that define the actions the user has to take, these requirements focus on the overall user experience, system performance, and operational standards. The critical system functions such as usability, scalability, security, and performance are taken care of by accommodating a diverse user base, securing user data, and making sure the app is efficient and accessible even with more users and data volume.

Performance

- The app should load the home screen and main functionalities within 1.5 seconds on an average smartphone.
- The recommendation algorithm must generate suggestions in less than 3 second after receiving the necessary input.

Scalability

- The app should be able to handle at least 50000 users concurrently without performance degradation.
- It should be designed to scale horizontally to accommodate future growth in user base and data volume.

Usability

- The app must be intuitive and easy to navigate, with a clean and responsive user interface.

Security

- User data, including preferences, reading history, and reviews, must be stored securely and protected against unauthorized access.
- The app must comply with relevant data protection regulations, such as GDPR, ensuring users' privacy.

Reliability

- The app should have an uptime of 99%, ensuring it is consistently available to users.
- Regular backups of user data and system settings must be performed to prevent data loss.

Maintainability

- The app's codebase should be well-documented and modular, allowing for easy updates and bug fixes.
- The system architecture should support future enhancements and integrations with minimal disruption to existing functionality.

Compatibility

- The app must be compatible with Android devices running version 8.0 (Oreo) and above.
- The app must be compatible with iOS devices running iOS 13 and above
- It should support various screen sizes, ensuring a consistent user experience across different devices.

3.6 System Constraints and Dependencies

The development and deployment of the book recommendation application involve several key system constraints and dependencies. These factors must be considered to ensure smooth implementation, integration, and performance. Below are the key constraints and dependencies associated with the project:

Backend Technology Stack

The backend of the application is built using Java and the Spring Boot framework. The choice of Spring Boot introduces certain dependencies on Java Development Kit (JDK) versions, particularly JDK 8 or later, which must be installed for the application to run in a PC. In the same manner, Spring Boot also requires the assistance of other tools and libraries like Spring Security for authentication, Spring Data JPA for database interactions, and other Spring modules. These dependencies need to be appropriately set up to ensure the backend works properly.

Database

The application's data is stored in a MySQL database. The application relies on MySQL database management for quick and efficient data retrieval and storage including user information, book data, reviews, and lists. It is necessary to have the right database indexing and query optimization so that the access to stored data is quick and reliable. Moreover, the database schema must be developed in a way that it can handle a massive amount of information as the application grows. It is required to have a compatible version of MySQL with the Spring Data JPA module and all the necessary database connectors must be installed in the backend environment.

Frontend Technology Stack

Flutter: The cross-platform front end of the application is entirely constructed with Flutter, which is the answer for both Android and iOS. The Dart programming language and several packages of the Flutter SDK are required for Flutter. The packages for UI design, state management, and networking such as HTTP for API requests are the main actors of the Flutter environment.

UI/UX Constraints: Since the front-end design focuses on a smooth user experience, Flutter's capabilities must be leveraged within its constraints, ensuring responsive and interactive user interfaces on various screen sizes.

AI Integration

OpenAI API: The effective working of AI-based book suggestions largely relies on the OpenAI API. This is especially for the purpose of creating recommendations that relate to the user's preferences. This sets up the need for an internet connection and could bring about costs by way of API usage volume directly related to the number of requests to OpenAI's servers.

Furthermore, the speed of the API is a factor that needs to be considered to make sure the app remains timely. On top of that, users are also likely to have rate-limiting issues subject to the number of usages allocated in the OpenAI subscription plan.

Operating System and Device Compatibility

When designing an app, there should be really careful thinking about the app's compatibility with Android and iOS, as Flutter is fully operative for OS variants. The quality of the app to ensure that it is flawless on the various devices and OS versions is crucial. This means that for Android developers, they have to take care of compatibility with new as well as old devices that probably cannot handle the full set of functionalities. Similarly, for iOS, the app must be optimized to run effectively on both recent and older iPhone and iPad models, ensuring a consistent and reliable user experience across different Apple devices.

External Libraries and APIs

The application is based on quite a few external libraries to operate correctly, which are different HTTP clients for network requests, JSON parsing libraries, and also authentication packages. Dependency management tools, like Maven for the backend and the package manager for the frontend in Flutter, are needed for the tasks of installing and updating of these libraries. Compatibility between library versions should be regularly monitored to avoid potential issues due to updates or deprecated features.

Performance and Scalability

The scalability of the backend and the database must be maintained in line with the number of users. This implies implementing regular database management, load balancing of backend services, and deploying hosting solutions that are capable of fluctuating levels of demand. Caching strategies may be employed to reduce database load for frequently requested data such as book recommendations

Network Constraints

The app's reliance on both the OpenAI API and database calls means that it requires a stable and reliable internet connection for optimal performance. Network latency and system interruptions have the potential to create a scenario of extreme dissatisfaction leading to negative customer experiences, especially when AI-solutions are involved. As a solution, local caching development and error connection management are recommended.

4 Design

The design chapter refers to the detailed structure and data management of a book recommendation application. A clear design will ensure that the application is running properly, treat it with a good response, and support scalability and maintainability. In this chapter, the design decisions with the system architecture, database structure, and overall organization of components are shown in detail. Through a firm foot as a base, the design is expected to ensure that users can communicate without any problems with the system, that it has easy data management and AI-driven recommendations, and that it can work most effectively on different devices.

Best practices in software engineering are chosen to frame the design process taking into account that the system should be modular, flexible, and allow future improvements. Database design is created so that it manages to store and retrieve books and users and review data in the shortest time possible, also maintaining the integrity and consistency of the data system. The design principles cited earlier combined together are essential for a robust and user-friendly application that is scalable and also supports book discovery.

4.1 System Design

The book recommendation application architecture is introduced by the layered client-server model, which is based on modularity, scalability, and maintainability. It is built to operate interactively, like user interactions, book recommendations, and data storage, based on the fact that concerns need to be separated clearly among the frontend, backend, database, and AI integration.

Frontend

The application's interface built on the Flutter framework is the first thing that the users come into contact with. Besides, they can explore the range of books, get recommendations based on their preferences, write reviews, and manage their library. Flutter makes it easy to deploy on different application platforms, such as Android and iOS.

- The frontend communicates with the backend via RESTful APIs, making HTTP requests to retrieve book data, submit user actions, and receive recommendations.
- It handles UI rendering, user authentication, input validation, and interaction with local storage when necessary.
- API responses from the backend are processed and displayed in a visually engaging and user-friendly manner

Backend

The Spring Boot-based backend serves as the core of the application, handling business logic, API requests, user authentication, and AI integration. It exposes a set of RESTful APIs that enable communication between the frontend, database, and AI services.

- It processes API requests from the frontend, validates input, and retrieves relevant data from the MySQL database.
- It implements user authentication and authorization using JWT (JSON Web Token) to ensure secure access to user-specific features.
- It forwards user preferences to the OpenAI API to generate book recommendations based on favorite books and processes the AI-generated responses before sending them back to the frontend.

Database

The MySQL relational database is responsible for storing and managing structured data, ensuring data integrity, consistency, and efficient retrieval. It is directly connected to the backend, which handles all database queries and updates.

- It stores user data (e.g., accounts, preferences, reviews, and reading history).
- It maintains a books table containing metadata such as titles, authors, genres, and summaries.
- Relationships between books and users (e.g., saved books, reviews, ratings) are managed through relational tables, allowing for efficient querying and filtering.
- ACID (Atomicity, Consistency, Isolation, Durability) compliance ensures that database transactions are reliably executed.

AI Integration

The OpenAI's API is used by the app to give unique book suggestions to users. The backend works as an intermediate between the front-end and the AI model by parsing the input and providing the proper responses to the front-end.

- When a user selects favorite books, the backend processes this input and sends it as a structured request to the OpenAI API.
- The OpenAI API analyzes the provided data and returns a consistent JSON response containing a list of "Title" values for book recommendations.
- The backend then validates and processes the AI-generated results before sending them to the frontend for display.

System Communication and Workflow

The components interact with each other in the following sequence:

1. The Flutter frontend sends an API request to the Spring Boot backend for user authentication, book data retrieval, or book recommendations.
2. The Spring Boot backend processes the request, interacts with the MySQL database if needed, and either returns the requested data to the frontend or proceeds to request recommendations from the AI.
3. For AI-generated recommendations, the backend formats a request and sends it to the OpenAI API.
4. The OpenAI API processes the request and returns a list of recommended books.
5. The backend validates and refines the AI response before sending the final recommendations to the frontend, where they are displayed to the user.

4.2 Layered Architecture

In order to attain modularity, maintainability, and scalability, the application uses a layered architecture. The structural layering of the codebase is the architectural pattern, where each layer is independent and deals with separate aspects of the system. Using such an organized method, the application achieves almost a perfect separation of tasks, which contributes to simplifications like being able to perform everything without having to interfere with the whole system. The primary layers in the architecture are the Presentation Layer (Controller), Business Layer (Service), Persistence Layer (Repository), and Database Layer (MySQL). Moreover, to acquire book suggestions, an AI is employed as an external service.

Presentation Layer

The Controller Layer is a response given to the request by the client, the main one from the Flutter frontend. It is developed in Spring Boot, and this layer is formed as a combination of numerous RESTful API endpoints that make it possible for the user to communicate with the backend. This is a unique division where the controllers are designed to satisfy the book retrieval, reviews, AI-generated suggestions, and the user authentication feature. The controllers interpret the incoming HTTP requests, validate the input data, invoke service-layer methods, and return the suited responses. So that the request route can be changed and the response can be organized in the multiple parts of the system, the Controller Layer manages structured and organized communication flow. [9]

Business Layer

The Service Layer is designed and integrated with Spring Boot Services, which is required to execute the system's business logic. It serves as an intermediary between the Presentation Layer and the Persistence Layer. This layer strictly enforces business rules and performs the necessary data conversions before interfacing with the database. The system remains modular and easy to maintain by separating business logic from the controller; thus, future changes can be made without affecting the API endpoints. [9]

Persistence Layer

The Persistence Layer is responsible for interacting with the MySQL database. In this layer, Spring Data JPA is used to execute queries, thereby ensuring that the system practically stores and retrieves data. User details, book lists, and reviews are some types of data that this system can read and write. The repository classes are a different layer of abstraction over direct SQL queries that guarantee optimal and secure database operations. The content of this later layer includes data that strengthens the integrity, such as data that makes sure the transactions are ACID (Atomicity, Consistency, Isolation, Durability) compliant. [9]

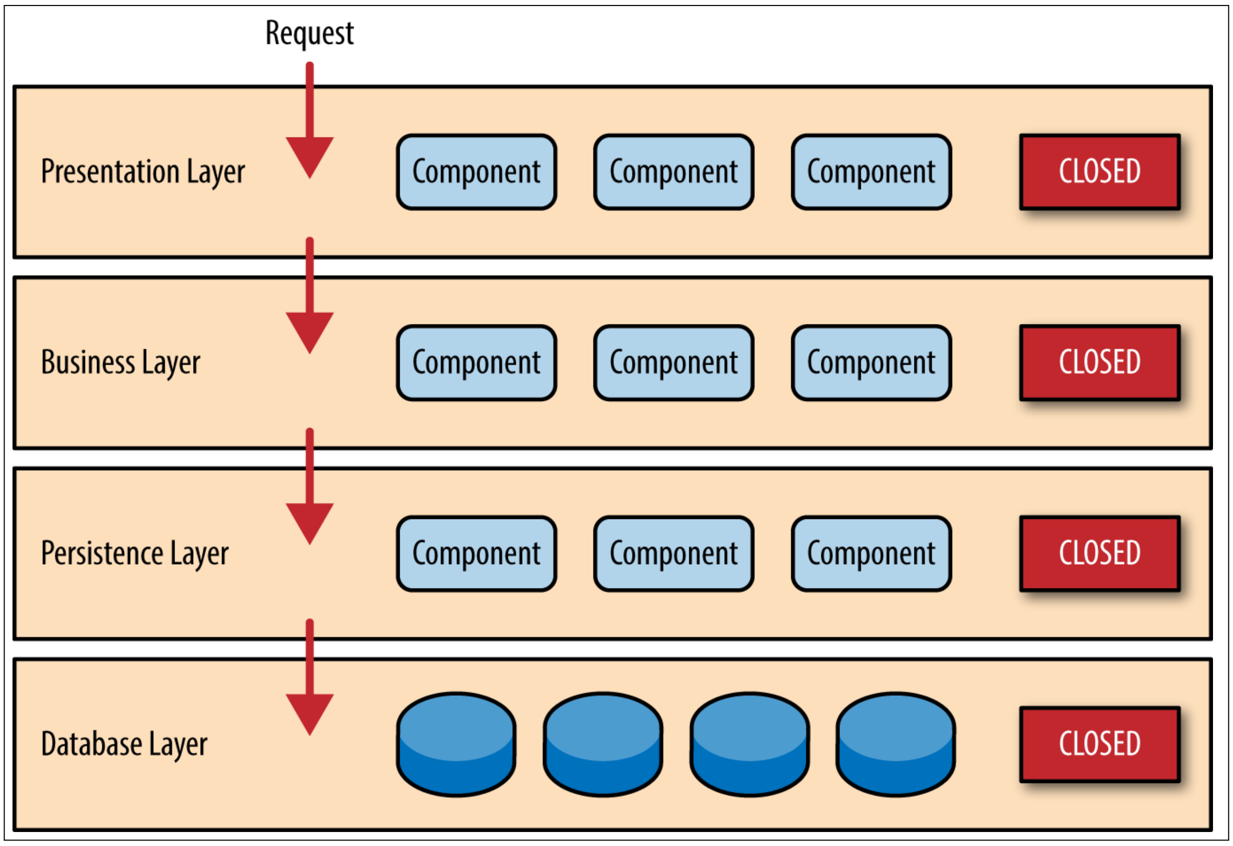


Figure 4.1: Layered Architecture Diagram [9]

4.3 Database Design

The backbone of the application's operation is the right choice of database architecture, which inwards drives efficiency, scalability, and integrity. The database that stores book recommendation data utilizes the Relational Database Model DBMS with MySQL as the management system. The goal is to be able to selectively and efficiently store and also manage user details, book details, reviews, lists, and ultimately, the interactions. The database schema is wisely designed to be able to maintain fast queries, data consistency, and scalability that would seamlessly fuse it with the backend. This philosophy, by which the database models the relationships between the entities, users, books, genres, ratings, and preferences, also enables fast and clean user data retrieval and management, thus enhancing the user experience.

The two largest and most important entities are "User" and "Book", which, just like their name shows, are the database tables that map the user and book details. The book table contains the most important information about a book, allowing for the link of this table to other important tables, e.g. Author. The User table, similarly stores the information about the user, allowing as well to link to other important tables, like User-Genre-list for example.

The other important entities are ”Genre”, ”Author”, and ”Review”, which contain the main information about respectively the genres, the authors, and the reviews that any user can leave for a book. These tables allow linking to many other tables, making them an important building block of the entire database

Then there are the entities created to create the link between the main entities. These connecting entities are namely:

- **Book-Genre:** This entity creates the link between the books and their genres. It contains the ID of the book and that of the genre. This way a book may be connected to many genres. Even though the book id may appear many times in this table, it is always connected to other genres.
- **Book-Author:** Similarly, this table links the authors to the books, where a book id is connected to only an author id, but a book id may appear many times, because a book can have many authors. Logically, also an author may appear many times, for an author may have written many books.
- **User-genre-list:** This table is used to represent the list of favorite genres for a specific user. It has the user ID as a field and an ID that is generated specifically for this list of this user. This ID creates a connection to the auto generated table named user-genre-list-genres, which represent the genres that are contained in the specific list of favorite genres created for that user.
- **User-author-list:** The same logic as for the list of genres exists also for the list of favorite authors. The auto generated table takes the ID of the specific list created by the user and links it to the authors that the user adds to the list.
- **User-Book-list:** Similarly to the list of genres and authors, this list is created for books. But there is a small difference: there are three kind of book lists, namely FAVORITES, NEXT, and READ, which represent fovorite books, books that the user would read next, and the books the user has already read. The kind of list is specified by an integer in this table where numbers 1, 2 and 3 are respectively FAVORITES, NEXT, and READ. After adding a book to a specific list, the id of that list is automatically generated, and a table named user-book-list-books is created, that connects that specific list to the books that are added to the list.

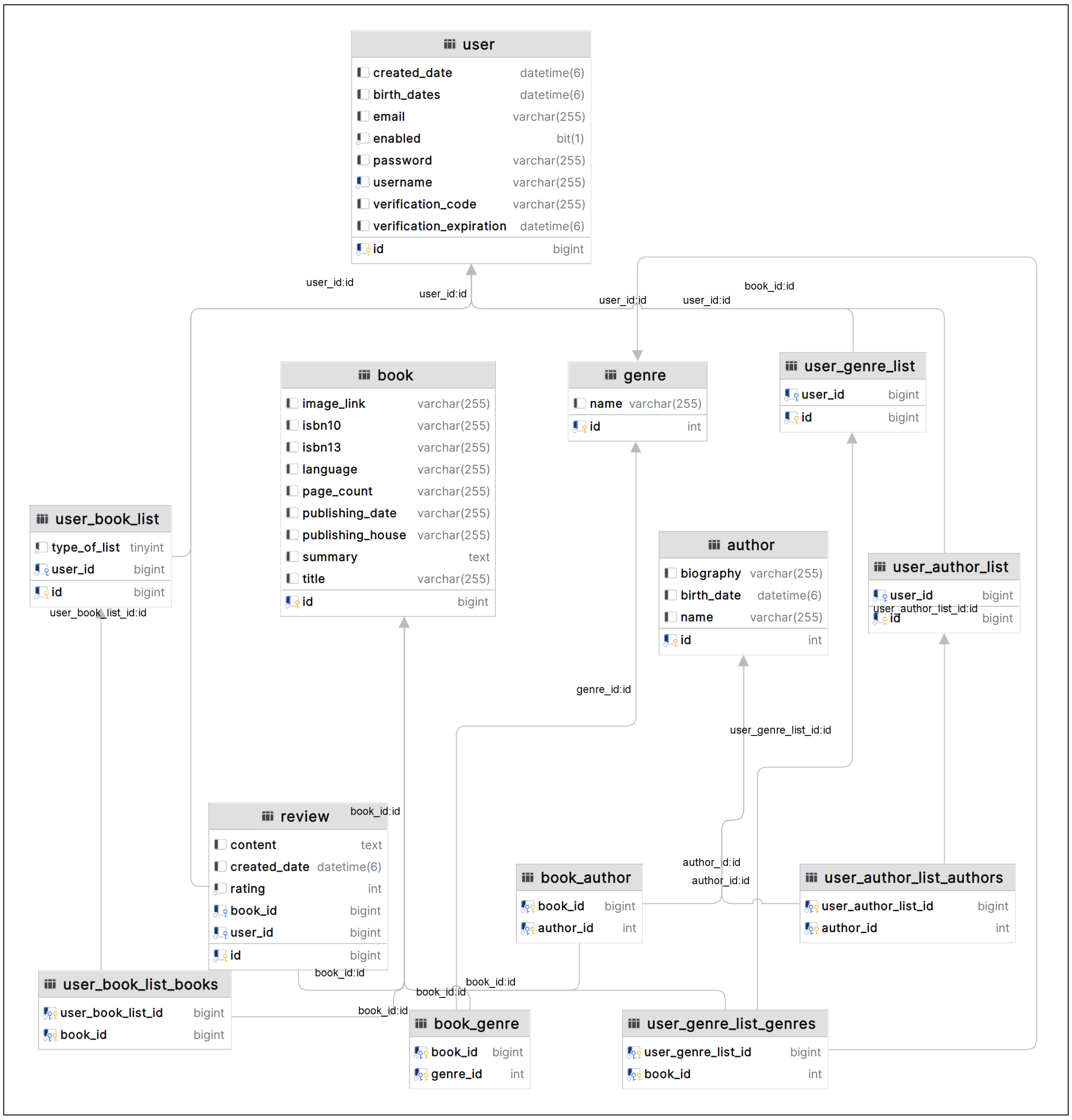


Figure 4.2: Database Design Diagram

5 Implementation

The implementation phase of the project is really about turning the architectural and database designs into a fully functional application. This step comprises the backend and the frontend development, in which high-level protocols are used to communicate between components so that efficiency, security, and scalability are maintained. Industry best practices and advanced technology are used to ensure that the system operates smoothly, is user-friendly, and can handle tough functionalities like book recommendations, user interactions, and AI-driven suggestions.

This chapter is a discussion of the stages of development. Therefore, the article explains in detail how Spring Boot is the framework utilized for creating RESTful APIs, the frontend development using Flutter, and the integration of AI features through OpenAI's API for book recommendations. Moreover, it issues a detailed account of the obstacles faced during development and what measures were taken to guarantee the system's reliability and efficiency.

5.1 Prerequisites

Before getting the application off the ground, it is necessary to configure the needed technologies and development environment. Apart from the fact that the system is constructed using a mix of backend, frontend, database, and AI components, where each element is based on a specific set of tools and frameworks, we must not bypass the influential part of ensuring the compatibility of these technologies, being the fact that this type of seamless integration is the only way out for top performance. Given below are the key requirements for the development of the application and its running:

Backend Environment

The backend of the application is created in Java 21 with the Spring Boot 3.3.3 framework. Java 21 is a long-term support (LTS) version, which provides better performance, stronger security, and new language features that help create an efficient backend. Spring 3.3.3 supports the building of REST APIs, dependency management, and proper integration of the databases and security mechanisms.

To build and run the backend in an effective way, the following tools are needed:

- IntelliJ IDEA 2024.1.2 (Ultimate Edition) – This IDE offers extraordinary support for Java and Spring Boot development and has the functionality of intelligent code completion, debugging tools, and integrated database control.
- Maven – As the main development tool, maven is used to handle dependencies and compile the project.
- Postman – Makes the testing of the API endpoints easy, and it also enables validation of the request-response structures and debugging of the backend functions

Database Requirements

The application was made to incorporate MySQL 8.0.37 as the relational database management system (RDBMS). MySQL is the preferred database for its flexibility, dependability, and the capability to focus on ACID transactions, which ensure data integrity and consistency. The database is built with Spring Data JPA that allows the Java objects to communicate with the database tables without hassles.

Frontend Environment

The frontend is developed using Flutter 3.24.4, a modern framework that enables cross-platform mobile application development with a single codebase for Android and iOS. Flutter is selected for its fast UI rendering, expressive widgets, and hot reload feature, which significantly speeds up the development process. The following tools are necessary for frontend development:

- Android Studio Koala | 2024.1.1 Patch 1 – The primary IDE for Flutter development, equipped with built-in support for Dart, Flutter widgets, and device emulation.
- Dart SDK – Required for writing and compiling Flutter applications.
- Flutter DevTools – Used for debugging, performance monitoring, and analyzing widget trees.

AI Integration and External APIs

AI-driven book recommendations are put into action by including OpenAI's API, which is implemented by using an API key that can be retreived from the OpenAI's official website. This API is used for the analysis of the user preferences and transmission of books in natural language understanding models. Communication between the OpenAI API and the backend is done using RESTful HTTP requests to make sure a well-structured and standardized response format is obtained.

Version Control

Version control is managed using Git, with repositories hosted on GitHub to facilitate collaboration and track development progress.

5.2 Backend Development

The backend forms the backbone of the application and is responsible for the business logic, data management, authentication, and AI integration. The backend of this app is developed in Java along with the Spring Boot framework. It is created in such a way that it is capable of scalability, security, and efficiency, such that the system can handle a large number of interactions at a time. The application is built on a RESTful architecture that allows the Flutter-based frontend and external services like the OpenAI API for book recommendations to communicate with each other without any problems.

JWT Security

The authentication system is a main component of the backend that carefully handles user privacy. JWT tokens are being used by the app for the purpose of authenticating the credentials, which are required to ensure secured access control. When a user registers or logs in, the backend issues a JWT token that is sent to the front-end, and from there, the token is included in the subsequent requests to get access to the protected resources. The approach described here is to ensure that actually only those users who are indeed authenticated can access - for example, saving books to their library or getting personalized recommendations.

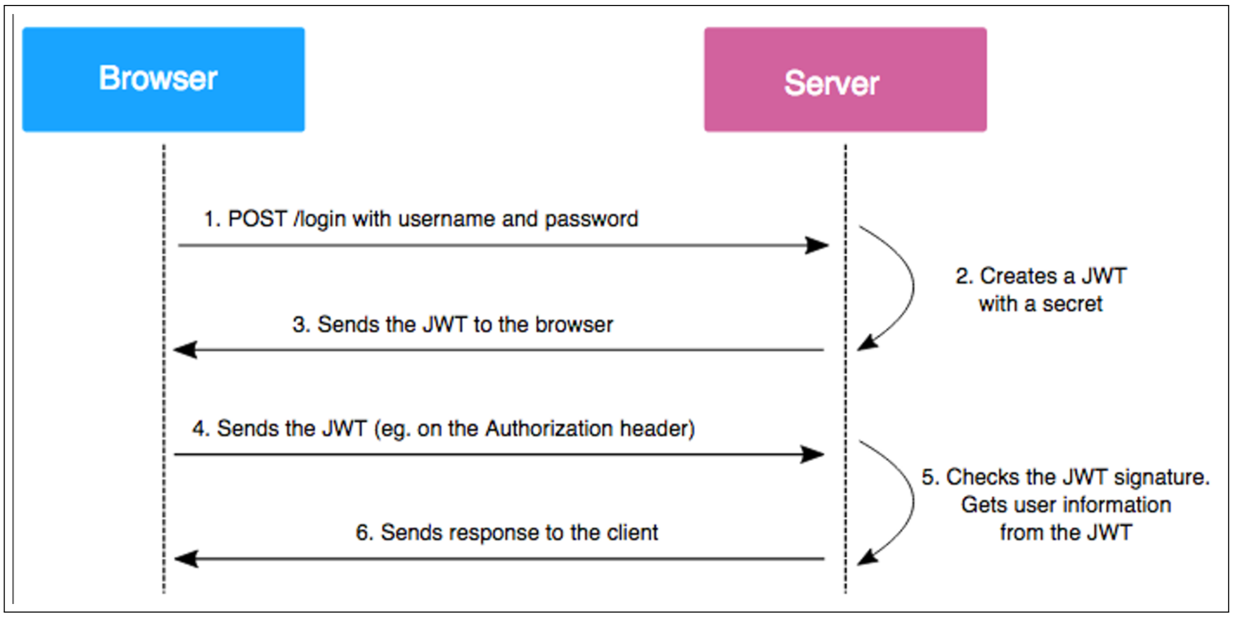

Browser
Server
1. POST /login with username and password
2. Creates a JWT with a secret
3. Sends the JWT to the browser
4. Sends the JWT (eg. on the Authorization header)
5. Checks the JWT signature. Gets user information from the JWT
6. Sends response to the client

Figure 5.1: JWT Sequence Diagram [10]

Spring Data JPA

Spring Data JPA handles the database layer more conveniently, which allows better interfacing with MySQL. Having the backend explicate the SQL database tables using the Entity classes and operating the CRUD (Create, Read, Update, Delete) operations through JPA repositories brings a level of simplicity. The organization is keeping a likeness in form with the users, books, genres, reviews, and favorite tables.

Not only does this structured approach go a long way toward guaranteeing that new functionality is easy to add, but it also significantly scales up the system's operational efficiency through JPA's notions of scalability, allowing it to work with raw SQL queries in a more straightforward way and still getting the flexibility to go very low level – if needed. Likewise, the use of lazy loading and pagination techniques facilitates the effective handling of massive data volumes, thus reducing the risk of performance issues related to query execution.

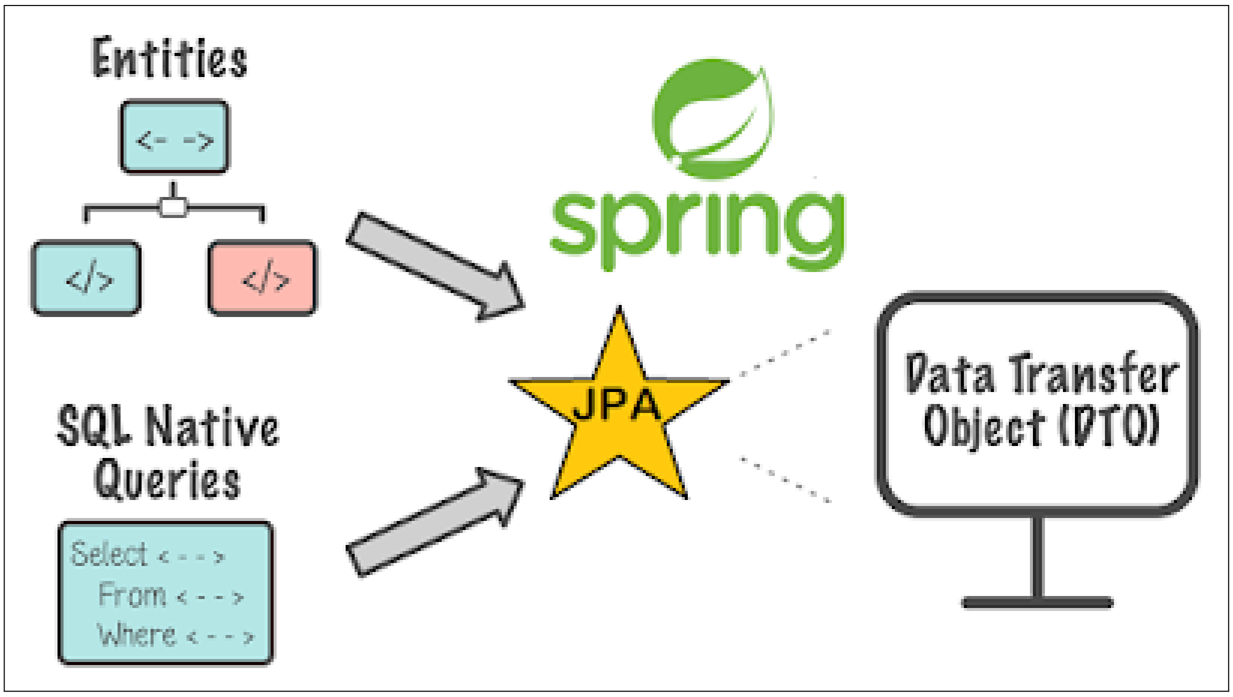


Figure 5.2: Spring Boot Data JPA [5]

AI Integration In Backend

A significant part of the back-end is the book recommendation system, which relies on the OpenAI API. Once the user chooses their favorite books, the back-end, through an API request, sends these data to the AI model. As a result, the back-end gets a structured list of recommended books, which is then processed and given back to the front-end to display. This process is aimed at being fast and dynamic, thus providing real-time recommendations that get better as the user's preferences change, ensuring real-time recommendations that improve as the user's preferences evolve.

Through Spring Boot ecosystem, the backend is developed to be modular and maintainable, which enables the future updates of easy feature expansion. The integration of RESTful APIs, JWT-based authentication, AI-powered recommendations, and an efficient database management system enables the creation of a sustainable, scalable, and high-performing backend architecture that is the mainstay of the application.

Class Diagram

A class diagram is used to represent the central elements of the application's backend, where the Controller and Service layers are in focus. These classes together form the core of the

RESTful API application, thereby ensuring the seamless communication of the Flutter frontend with the Spring Boot backend. With a model based on a layered architectural structure, the system is able to deliver modularity, maintainability, and scalability.

In the **Controller layer**, classes play the role of processing HTTP requests from the frontend. Every controller is responsible for endpoints that enable different functions like a user's login, book management, and AI-driven book recommendations. Thus, these controllers play a role as intermediaries, which are in charge of business logic and, at the same time, make a query to the underlying Service layer for the request validation and response formatting.

The **Service layer** is central to the function of the system. It manages the data and talks to external APIs, e.g., the OpenAI API for book recommendations. The service classes verify that the data methods will be efficient and consistent before returning it to the controllers. Thanks to the separation of concerns between the controllers and services, the system can develop a sustainable architecture which is easy to manage and extend.

This diagram provides a high-level view of how these components interact, highlighting the structured flow of data and functionality throughout the backend.

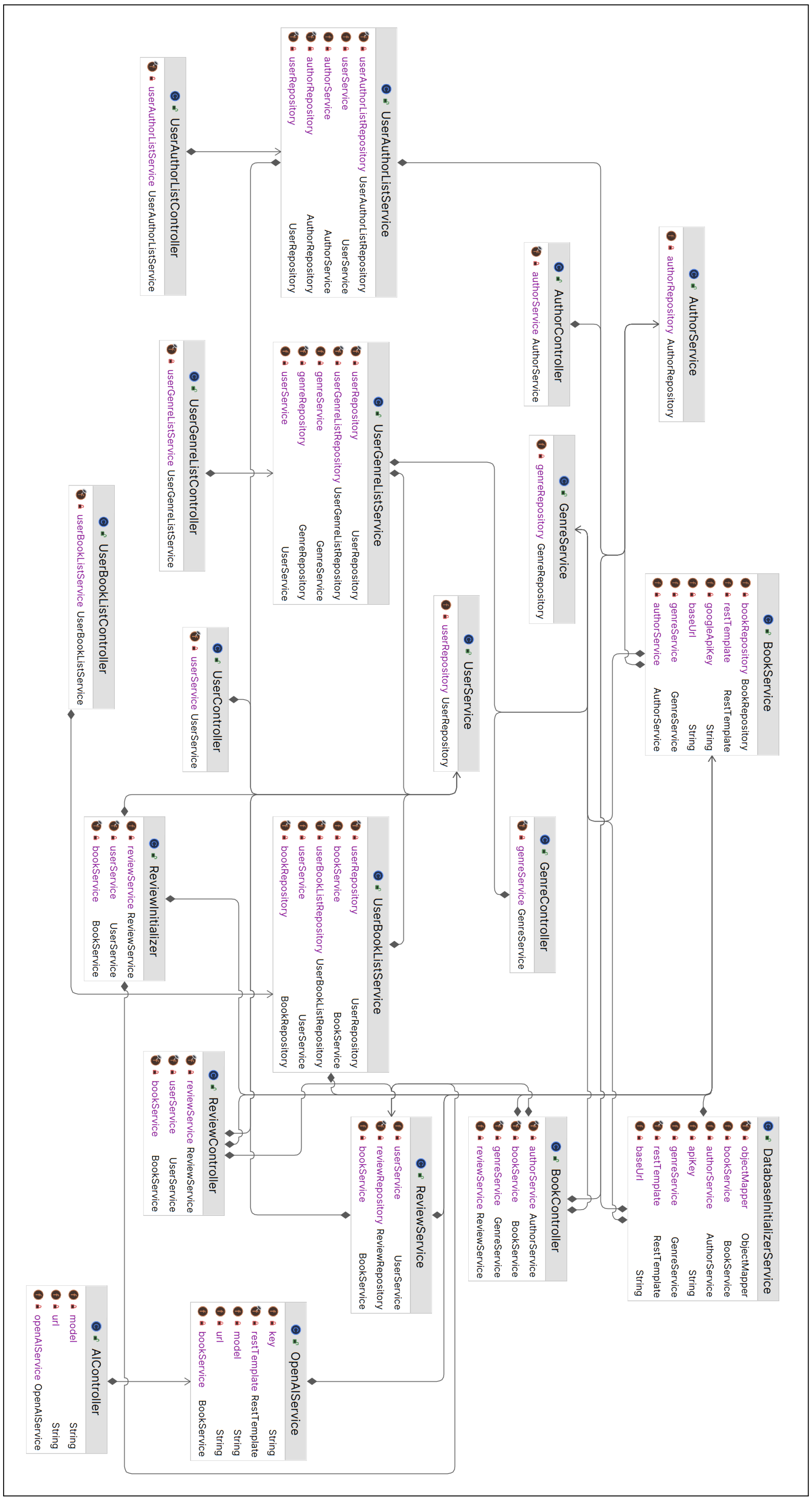


Figure 5.3: Class Diagram of Controller and Service layers

5.3 Frontend Development

Flutter, a cross-platform mobile app development framework developed by Google, is the basis of the frontend part. It can be used on any device, and developers only have to write code once for the app to work seamlessly on both Android and iOS. The power of Dart, Flutter's programming language, provides good performance, fast response, and smoothness that are attractive to the eye. This frontend module is strategically the best means of interaction because it implements smooth flow, has interactive UI components, and is able to handle communication using HTTP REST with the backend.

One of the main features of Flutter is the widget-based architecture. This feature gives an advantage because it makes a straightforward system to use. The user interface can be designed through small pieces of software, which are easily modifiable.

A user interface design, which applies Material Design principles, is used to help develop a consistently user-friendly experience across all devices. ListView, GridView, and Card are some of the built-in widgets in Flutter that are used in the book listings. Besides those, interactive elements like sliders, buttons, and text fields are added for user engagement. Moreover, animations and transitions are made for a streamlined navigation experience. Hence, the book discovery and interaction are more appealing.

API Integration

Frontend communicates with the Spring Boot backend over HTTP Interface. By enabling proper authentication headers, each API request is composed of secure communication. A few of the major API actions are as follows:

- User Authentication: Sending signup, login, and verification requests to the backend.
- Fetching Book Recommendations: Sending user preferences to the backend that, in turn, calls the OpenAI API for book suggestions.
- Managing User Libraries: Allowing users to add and remove books from categories such as Favorites, Next, and Read, as well as storing these preferences in the database.
- Submitting and Viewing Reviews: Allowing users to write and view book reviews, with the frontend dynamically updating based on new submissions.

Optimizing for Cross-Platform Performance

Being a cross-platform application means that it needs to work perfectly in both Android and iOS in order to guarantee the top quality performance. This involves:

- Adaptive UI: Creating a responsive user interface flow for different screen sizes.
- Platform-Specific Adjustments: Modifying platform-specific differences like gestures, status bars, and native navigation behaviors to let the app work seamlessly on both Android and iOS devices.
- Efficient Asset Management: Smoothing images, icons, and animations to increase the loading speed and lessen memory usage through enhancing compressing technology.

Enhancing User Experience

Moreover, the frontend has been further improved for ease of use so that the users can search for books, authors or genres, and rate the books system with a friendly interface. Furthermore, in the following versions, the users will be easily updated with any new information about books and community activities through the system of push notifications that can also be implemented.

Forging an application in Flutter is quick and easy, and it opens up a wide array of interface tools and makes it cross-platform software. Thus, it secures a very active and smooth user experience and is also one of the most efficient ones. Besides, the well-defined API-based architecture contributes to the app's functioning by providing an immediate connection between the user and the backend, which enables smooth, interactive, and real-time handling.

5.4 Integration of AI

The implementation of artificial intelligence (AI) in the application is a crucial part of the recommendation process. The traditional recommendation systems are characteristically based on collaborative filtering or content-based filtering, whereas this application uses OpenAI's API to offer dynamic, personalized book suggestions that are chosen on the basis of the user's preferences.

AI-Powered Recommendation System Workflow

The backend, built with Spring Boot, acts as the intermediary between the Flutter frontend and OpenAI's API. The recommendation process follows these key steps:

- User Input Collection: When a user selects their favorite books and asks the app for suggestions, the frontend sends this data to the backend via a RESTful API request.
- AI API Request: The backend structures this data and forwards it to OpenAI's API, utilizing ”function calling” to request book recommendations in a consistent JSON format.
- Processing AI Response: The AI returns a list of recommended book titles. To enrich these results, the backend matches them against locally stored book data in the MySQL database or retrieves additional details from an external book API (such as author, cover image, and summary).
- Data Transformation: The retrieved book data is mapped into BookDTO objects to ensure a structured response format.
- Displaying Recommendations: The backend sends the final structured response to the frontend, which then renders detailed book cards displaying the recommendations in an intuitive and visually appealing way.

To achieve relevance and adherence in the recommendations, the system adopts OpenAI's ”function calling” mechanism. By introducing a structured function for book recommendations, the API is programmed to produce a standard JSON format that is used to store book titles as responses. This method avoids unnecessary variability in AI response, and the back-end will be able to comfortably and quickly extract the relevant book data. As the AI provides book titles only, the back-end accordingly finds the details of matching books in the database.

When the structured recommendations are received, the back-end maps the returned book data to BookDTO objects before delivering. This ensures that the user is presented not only with textual recommendations but the whole book object, which contains additional information such as author, cover image, and description. By using this structured approach, the recommendation system becomes very accurate and, on top of that, gives a very friendly interface to the user so he can easily find new personal book recommendations.

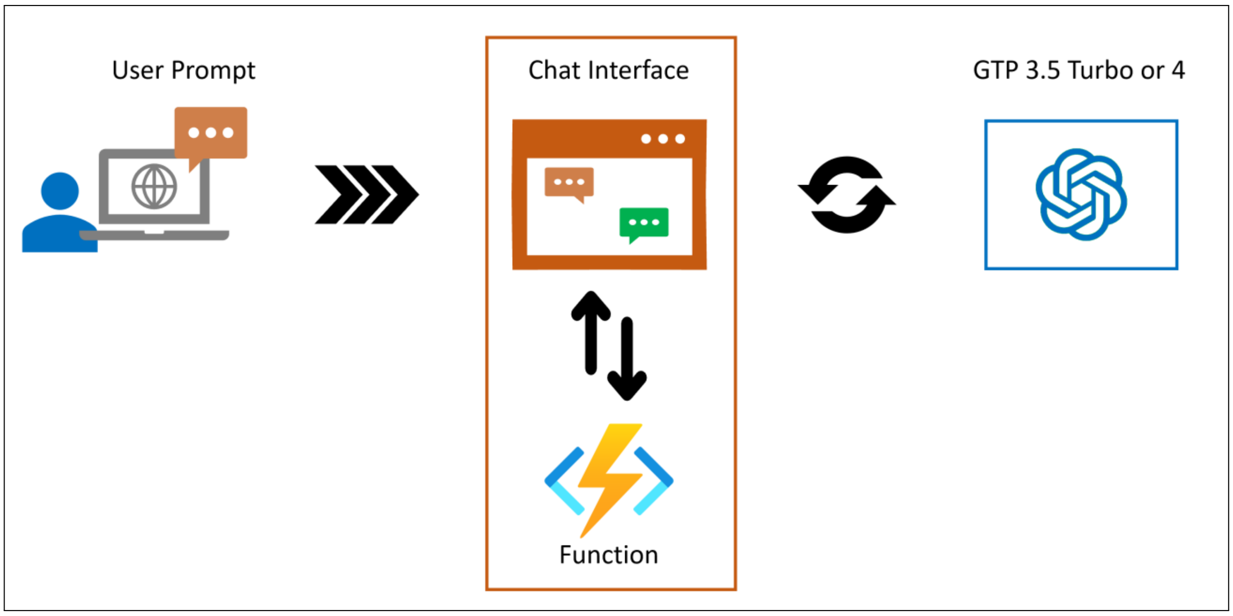


Figure 5.4: OpenAI's Function Calling Diagram [13]

Enhancing User Experience Through AI

With AI-powered suggestions the application incorporates the following features:

- Personalized Discovery: Instead of general lists based on how books are popular, users get lists that fit their needs.
- Scalability: The recommendation engine becomes more adept at users' changing preferences and their capability of processing larger data sets.
- Real-Time Processing: By optimizing API calls and response handling, the system provides fast feedback, ensuring a smooth user experience.

This AI-powered approach significantly enhances book discovery, allowing users to explore titles they might not have encountered otherwise. By integrating machine learning principles into the recommendation process, the system becomes more intelligent, adaptive, and user-centric, elevating the overall experience of book enthusiasts.

5.5 Summary of Code Functionality

The Java code snippet is responsible for making an API request to OpenAI to generate book recommendations based on user-provided favorite books. It constructs a request payload, defines the function-calling mechanism, and sends an HTTP POST request using Spring Boot's RestTemplate.

Constructing the AI API Request Body

A HashMap is created to hold key-value pairs for the request and the OpenAI model is specified. The temperature parameter is set to 1.3, influencing response randomness. The messages list is populated with a system message defining the assistant's role as a book recommendation assistant and a user message containing the list of favorite books and requesting five new, similar books without duplicating the provided ones.

```java
Map<String, Object> requestBody = new HashMap<>();
        requestBody.put("model", model);
        requestBody.put("temperature", 1.3);
        requestBody.put("messages", Arrays.asList(
                Map.of("role", "system", "content",
                        "You are a book recommendation assistant. Your task is to
                            recommend similar books."),
                Map.of("role", "user", "content",
                        "Given these favorite books: " + String.join(", ",
                            favoriteBooks) +
                                ", suggest 5 other books that are similar to these. Do
                                    not include the provided books in the results.")
        ));
```

Listing 5.1: Constructing the API Request Body

Defining Function Calling for Structured Output

- A custom function named "suggest_other_books" is created to enforce structured AI responses.
- It specifies that the AI must return an array of exactly five book titles, ensuring consistency.
- The structure is defined using JSON Schema, requiring the response to contain a titles field (a list of five book titles).

In order to make the AI return concise and structured recommendations, a "suggest_other_-books" function has to be defined. It contains the AI's directions to return only five titles in a JSON format. The process is completed when the JSON Schema specifies that the "titles" field has to be an array of five strings. This way only, the response structure remains the same. The function calling is the way to make sure that the book suggestions are received in the format that is expected.

```
List<Map<String, Object>> functions = List.of(
        Map.of(
            "name", "suggest_other_books",
            "description", "Suggest 5 other books.",
            "parameters", Map.of(
                "type", "object",
                "properties", Map.of(
                    "titles", Map.of(
                        "type", "array",
                        "items", Map.of("type", "string"),
                        "minItems", 5,
                        "maxItems", 5,
                        "description", "A list of 5 book titles similar to the input
                            but not matching them."
                    )
                ),
                "required", List.of("titles")
            )
        )
    );
    requestBody.put("functions", functions);
```

Listing 5.2: Defining Function Calling

Setting Up and Sending the HTTP Request to AI

- Authorization headers are set using a Bearer token.
- The request is wrapped in an HttpEntity and sent via restTemplate.exchange(), making an HTTP POST request to the specified url.
- The API response is received as a ResponseEntity<Map>, which can then be processed further.

```java
HttpHeaders headers = new HttpHeaders();
        headers.set("Authorization", "Bearer " + key);
        HttpEntity<Map<String, Object>> entity = new HttpEntity<>(requestBody, headers)
            ;

        // Send the request and get the response
        ResponseEntity<Map> response = restTemplate.exchange(url, HttpMethod.POST,
            entity, Map.class);
```

Listing 5.3: Sending the HTTP Request

Extracting Book Titles from the AI Response

The getBooksFromResponse method is responsible for parsing the API response and extracting the recommended book titles. Since the OpenAI API returns a structured JSON response, this method ensures that the relevant data is correctly retrieved and formatted for further processing.

- **Processing the API Response:** Upon receiving the API response wrapped in a ResponseEntity<Map>, the method first checks whether the response body is non-null. Then it extracts the ”choices” list, which contains different response options generated by the AI. The implementation selects the first choice (index 0), because it holds the most relevant information (based on the OpenAI response type). Within this choice, the method retrieves the ”message” object, which contains the ”function_call” field.
- **Parsing the Function Call Arguments:** If a function call exists in the response, it contains an ”arguments” field, which is a JSON-formatted string holding the list of suggested book titles. To process this, the method uses Jackson’s ObjectMapper to parse the JSON string into a Map<String, Object>. From this parsed object, it retrieves the ”titles” array, which holds the book recommendations.
- **Returning the Extracted Titles:** Once successfully parsed, the method stores the recommended book titles in a list (titleList) and returns it. This allows the backend to further process the recommendations, such as matching them with additional book details from the database before sending them to the frontend for display.

```java
private List<String> getBooksFromResponse(ResponseEntity<Map> response) {

        List<String> titleList = new ArrayList<>();
        if (response.getBody() != null) {
            List<Map> choices = (List<Map>) response.getBody().get("choices");
            if (!choices.isEmpty()) {
                Map choice = choices.get(0);
                Map message = (Map) choice.get("message");
                Map functionCall = (Map) message.get("function_call");
                if (functionCall != null) {
                    // Extract the "arguments" string (which is in JSON format)
                    String argumentsString = (String) functionCall.get("arguments");
                    try {
                        // Parse the JSON string into a Map
                        ObjectMapper objectMapper = new ObjectMapper();
                        Map<String, Object> arguments = objectMapper.readValue(
                            argumentsString, Map.class);

                        // Extract the "titles" array from the parsed Map
                        List<String> titles = (List<String>) arguments.get("titles");
                        titleList.addAll(titles);

                    } catch (Exception e) {
                        // Handle the exception if parsing fails
                        e.printStackTrace();
                    }
                }
            }
        }
        return titleList;
    }
```

Listing 5.4: Extracting Book Titles From Response

Mapping Book Entities to DTOs

The returnListOfBookDTO method plays a crucial role in transforming the list of book titles retrieved from the AI recommendation system into detailed book objects. This method takes a list of book titles as input and retrieves the corresponding full book details by invoking bookController.getBooks(titleList). This controller method is responsible for querying the database to find books that match the recommended titles.

```java
private ResponseEntity<List<BookDto>> returnListOfBookDTO(List<String> titleList) {
        List<Book> books = bookController.getBooks(titleList);
        List<BookDto> bookDTOs = books.stream()
                .map(book -> new BookDto(
                        book.getId(),
                        book.getTitle(),
                        book.getImageLink(),
                        book.getAuthors().stream()
                                .map(author -> new AuthorDto(
                                        author.getId(),
                                        author.getName()
                                )).collect(Collectors.toSet()),
                        book.getGenres().stream()
                                .map(genreDto -> new GenreDto(
                                        genreDto.getId(),
                                        genreDto.getName()
                                ))
                                .collect(Collectors.toSet()),
                        book.getIsbn10(),
                        book.getIsbn13(),
                        book.getPublishingDate(),
                        book.getPublishingHouse(),
                        book.getLanguage(),
                        book.getPageCount(),
                        book.getSummary()
                ))
                .collect(Collectors.toList());
        return bookDTOs.isEmpty() ? ResponseEntity.noContent().build() : ResponseEntity
            .ok(bookDTOs);
    }
```

Listing 5.5: Converting Book Titles into Detailed Book DTOs

In order to assure clean API responses and prevent the misuse of database entities, the approach changes the received Book objects into BookDto classes with the help of the Java Stream API. A single object is formed for a single book that contains all the essential fields required, like title, image link, ISBNs, and summary. Additionally, nested relationships such as authors and genres are mapped to their respective DTO representations (AuthorDto and GenreDto) to maintain structured and modular data. The method then returns a ResponseEntity, ensuring that if no books are found, it responds with a 204 No Content status. Otherwise, it sends a 200 OK response with the list of book DTOs. By applying this structured approach, it is guaranteed that the frontend will receive the information about the book entirely and in the correct format. Thus, the user experience at the time of the recommendation display will be improved.

5.6 Challenges Encountered and Solutions Applied

Developing a cross-platform book recommendation application with AI-driven suggestions presented a variety of challenges across backend development, frontend integration, AI interaction, and database management. These challenges required strategic solutions to ensure a scalable, efficient, and user-friendly experience.

Challenge 1: Handling AI Response Variability

One of the main difficulties faced was making sure that OpenAI API returned highly structured and regular outputs. Large language models (LLMs) tend to frequently create texts in a free-form manner. Thus, there might be discrepancies in how the titles of books were returned, which would be very hard to map to real-life book records stored in the database.

Solution: Using OpenAI Function Calling

Backend uses OpenAI's function calling feature to enforce a structured JSON response to solve this. The approach is to return a set of book titles in a pre-implemented schema instead of unstructured text. Furthermore, this approach guarantees the backend can process, match, and enhance the book selection with metadata from MySQL or the external book API in a dependable manner.

Challenge 2: Asynchronous API Calls and UI Responsiveness

Flutter applications rely heavily on asynchronous operations, particularly when making network requests. Calling different back-end APIs can introduce noticeable loading times, which might negatively impact the user experience if not handled properly.

Solution: Implementing Efficient State Management and Loading Indicators

Flutter client-side uses state-management solutions to keep the user interface stable so that the UI is not frozen or blocked during the API calls. Another advantage of these intermittent recommendation fetches is that the loading indicators and the skeleton screens demonstrate to the user that the system is working and provide visual cues in place of the UIs which remain empty or even stagnant.

Challenge 3: Populating the Database

In order for the program to works seamlessly, many books should be included in the database. This way, when the AI suggests books, the backend can actually find them in the database and send them to the frontend.

Solution: Google Books API

For this system, the approach to database population is the use of Google Books API. First, a specific number of books is fetched from the Google Books dataset, at least as much as the free API calls allow. Then, if the AI suggests a book that is not in the database, the backend notices that and makes a call to the Google Books API in real-time, fetches the book, and puts it in the database of books. Then the suggested books from the AI are sent to the frontend.

Challenge 4: Ensuring Consistent User Experience on Android and iOS

While Flutter simplifies cross-platform development, variations in device capabilities, OS versions, and screen sizes can lead to inconsistencies in UI behavior, navigation flows, and performance.

Solution: Adaptive UI Design and Platform-Specific Optimization

By leveraging Flutter's responsive layout techniques, the UI adapts to different screen sizes dynamically. Platform-specific optimizations, such as handling iOS's default navigation gestures or optimizing animations for lower-end Android devices, ensure a smooth experience across all devices.

6 Demonstration

This chapter demonstrates the functionality of the introduced application by a series of screenshots and explanations. The purpose of this demonstration is to showcase how different characteristics of the application are being applied, showing the user experience of the application from navigation and other core functionalities such as book discovery, recommendations, and reviews. Each section will emphasize the main interactions, explaining how users interact with the system and how the backend processes these interactions to deliver precise results. This chapter uses real examples extracted from the app to make it possible for the readers to realize how the implemented features combine to form a seamless and personalized reading experience.

6.1 User Authentication

Authentication is an essential factor in the working of the application. It assures the safety of the user accounts and, at the same time, delivers the login experience. The authentication process is divided into three key procedures: Signup, Email Verification, and Login. These steps put together help to ensure the user accepts access to modified features, which are recommendations to books and saved preferences.

Sign Up and Verification

The signup screen enables a new user to set up their account by typing in their email, their username, and password. The backend of the system sends a verification code to their email and asks for this code in the Verification screen. If the correct code is provided by the user, the data is registered in the database by the backend, and thus, the account is created successfully. This shows that the service allows only the proper email address to be used for account creation, which in turn prevents intrusions and increases security.

Log In

After the verification, the users can use the app using their credentials. The login screens demand the input of the email and password of the user, which is then verified against the database. Subsequently, the backend produces a JWT (JSON Web Token) which is used for secure access to the app's features. So, the users can stay logged in every time they come back, as their session is always protected by token-based authentication.

These authentication mechanisms provide a balance between security and user convenience, making sure that the system remains safe while offering a smooth onboarding experience. The provided screenshots illustrate the signup, verification, and login processes within the app.

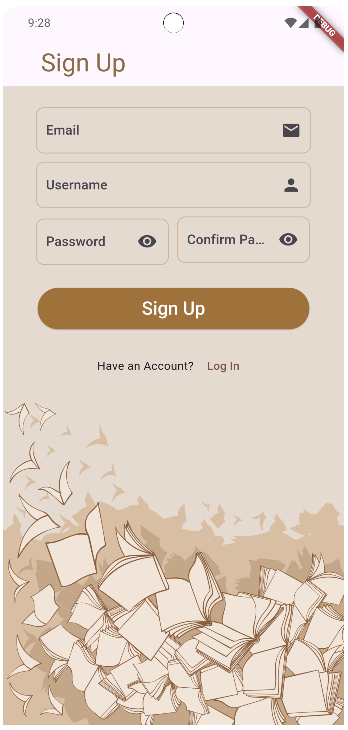


Figure 6.1: SignUp Screen

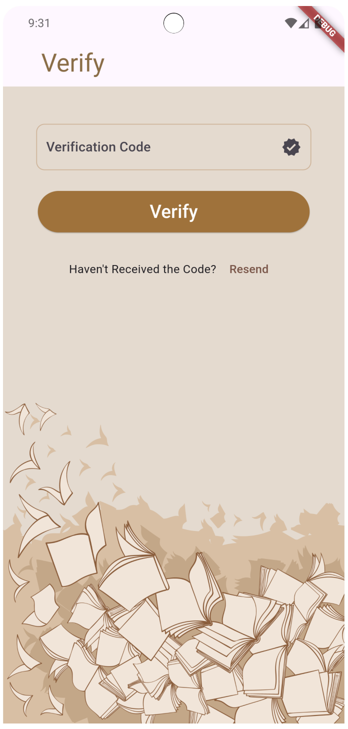


Figure 6.2: Verification Screen

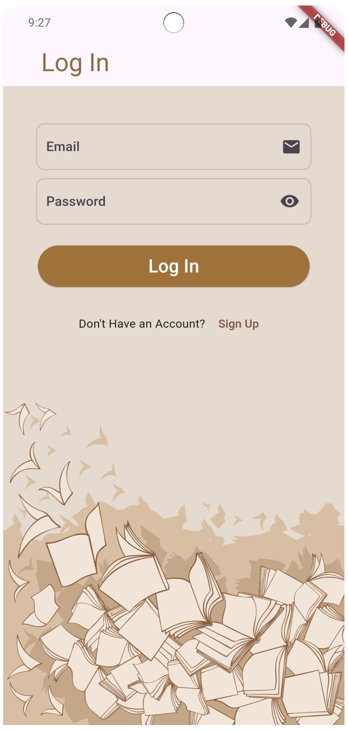


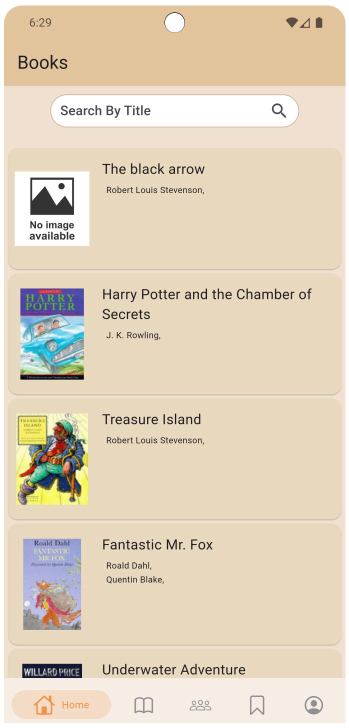
Figure 6.3: LogIn Screen

6.2 Main Screen

The main screen is divided into five pages: Home, Genres, Authors, Library and Profile.

6.2.1 Home

At the very beginning of the app, the Home page is shown. On this page, a list of the top-rated books is displayed. This rating is based on the user ratings of all the books and specifically in this application. The ratings are collected by the backend from the database, and the list of the top-rated books is generated. Then, the list is sent to the frontend using DTOs, so the frontend reads a book as a book card and thus can show all the details of the specific book.
There is also a search bar that allows the user to search a book by the title. This works by retrieving all the books that contain that part of the title from the database.



The black arrow
Robert Louis Stevenson,

Harry Potter and the Chamber of Secrets
J. K. Rowling,

Treasure Island
Robert Louis Stevenson,

Fantastic Mr. Fox
Roald Dahl,
Quentin Blake,

Underwater Adventure

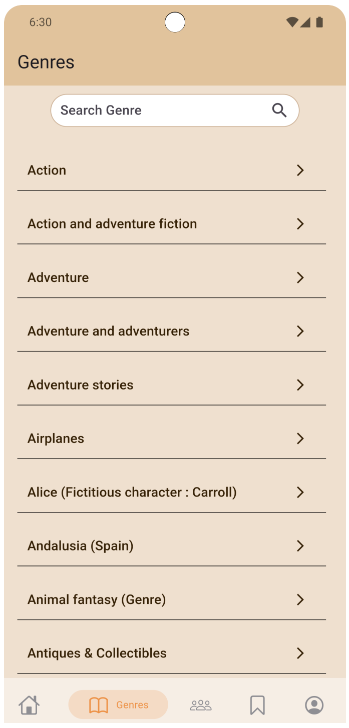
Figure 6.4: Home Screen

6.2.2 Genres

The second page, shown at the bottom navigation bar by an open-book figure, is the Genres page. Here, a list of all the genres is shown. The list is generated by the backend after retrieving it from the database, and the frontend displays this list of all the available genres. The genres are alphabetically ordered, and there is also a search bar at the top of the screen. Via this search bar, the user can search a specific genre, thus being able to click that genre and see the books that belong to it.

When the user opens a specific genre, all the books categorized in that genre are listed. This way, the application helps the users find exactly what they are looking for.

Another important point on the screen of a specific genre is the option to add the genre to the user's favorites. In the top-right, there is a bookmark sign and a text that reads ”Add to Favorites.” When the bookmark sign is filled, it means that the genre is already in the user's favorite genres list. Likewise, when the bookmark sign is empty, the genre is not in the list, but it can be added by tapping that sign.



6:30

Genres

Search Genre

Action

Action and adventure fiction

Adventure

Adventure and adventurers

Adventure stories

Airplanes

Alice (Fictitious character : Carroll)

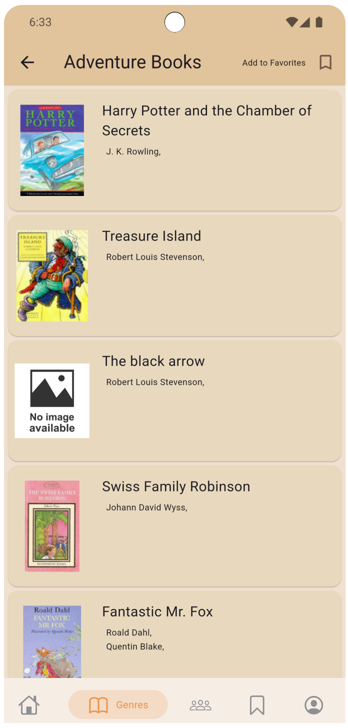
Andalusia (Spain)

Animal fantasy (Genre)

Antiques & Collectibles

Genres

Figure 6.5: Main Genres Screen



6:33

Adventure Books

Add to Favorites

Harry Potter and the Chamber of Secrets
J. K. Rowling,

Treasure Island
Robert Louis Stevenson,

The black arrow
Robert Louis Stevenson,

No image available

Swiss Family Robinson
Johann David Wyss,

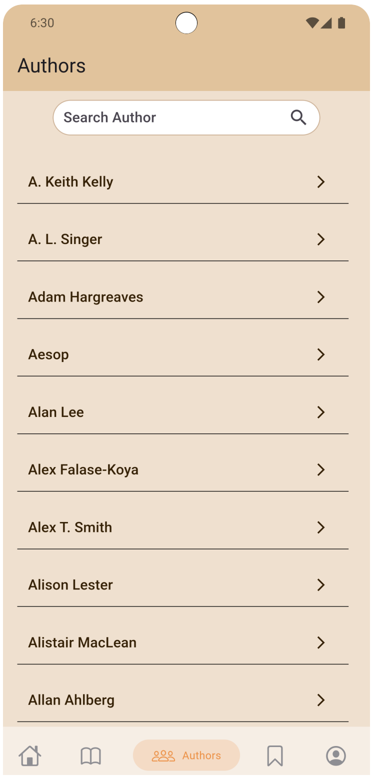
Fantastic Mr. Fox
Roald Dahl,
Quentin Blake,

Genres

Figure 6.6: Specific Genre Screen

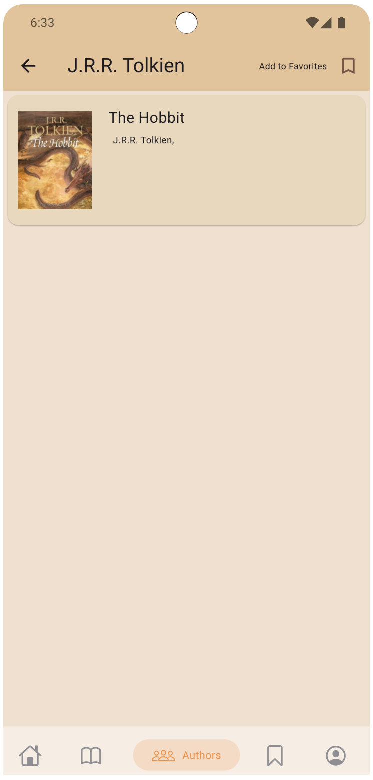
6.2.3 Authors

The third page, accessible from the bottom navigation bar via a three-people icon, is the Authors page. This section displays a list of all authors, which the backend retrieves from the database. The frontend presents the authors in alphabetical order, making it easy for users to browse. A search bar at the top allows users to quickly find a specific author and view the books they have written. Selecting an author opens their dedicated page, where all their books are listed. This feature helps users explore books by their favorite writers effortlessly. Additionally, users can add authors to their favorites. A bookmark icon in the top-right corner, accompanied by an ”Add to Favorites” label, lets users save their preferred authors. If the icon is filled, the author is already in the favorites list; if empty, tapping it will add them.



A. Keith Kelly
A. L. Singer
Adam Hargreaves
Aesop
Alan Lee
Alex Falase-Koya
Alex T. Smith
Alison Lester
Alistair MacLean
Allan Ahlberg

Figure 6.7: Main Authors Screen



The Hobbit
J.R.R. Tolkien,

Figure 6.8: Specific Author Screen

6.2.4 Library

The fourth page, displayed on the bottom navigation bar by a bookmark symbol, is the Library page. Here, the different lists are shown. There are three lists for books: Favorites, Want To Read, and Read. Just like the names suggest, these lists represent the favorite books of the user, the books that the user wants to read next, and the books already read by the user. A list named My Authors shows the favorite authors that the user has inserted as their favorites, and the same logic for My Genres, where the user can see the favorite bookmarked genres.

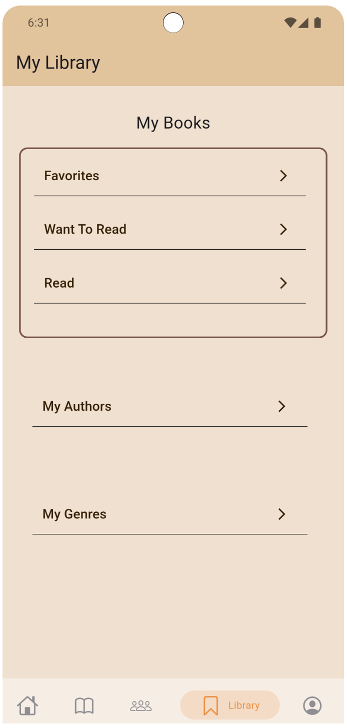


Figure 6.9: Library Screen

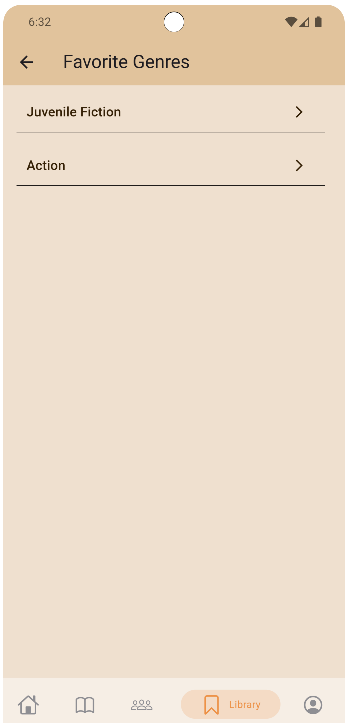


Figure 6.10: Favorite Genres

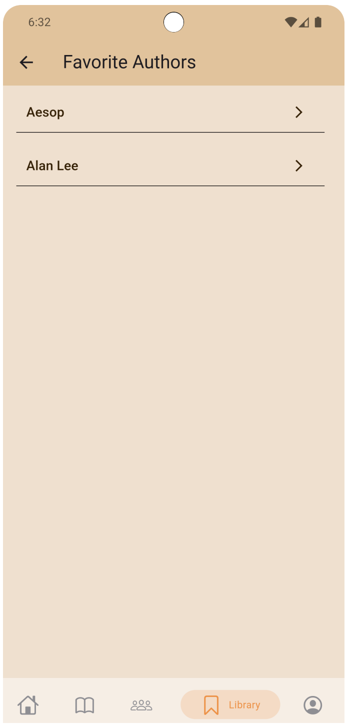


Figure 6.11: Favorite Authors

6.3 Favorite Books and AI Suggestions

The Favorite Books screen is a critical element in guiding the personalized recommendation system of the application. It allows users to set up what books they like while being assisted by AI in getting custom-tailored book suggestions. That being said, the application of AI ensures that the suggestions are actually driven by the user's personal reading preferences, hence enriching their book discovery experience.

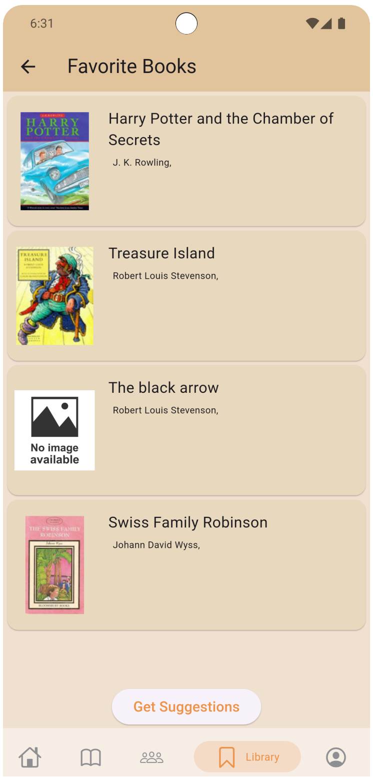
Selecting Favorite Books

Adding books to the Favorite Books in an individual book screen is the starting point for AI-generated relevant recommendations. These are all saved in the database, and the engine takes them as input for the recommendation system. Thanks to this feature, the system can learn the user's reading preferences in a better way and suggest books that will stand near to their taste.

Generating AI-Powered Book Suggestions

A ”Get Suggestions” button is quite noticeably displayed on the screen, therefore users can have a whole list of suggestions with just one touch. After the button is pressed, the app transmits the detailed list to the backend as a list of strings containing only the titles of the favorite books. The backend then communicates with OpenAI’s API and fetches five book ideas for these favorite books.

To ensure high-quality and consistent responses, the backend uses the function calling feature of OpenAI, which instructs the AI to return structured JSON responses containing only a list of book titles. These titles are then cross-referenced with existing book data in the database, allowing the system to retrieve full book details and thus send them to the frontend for display.



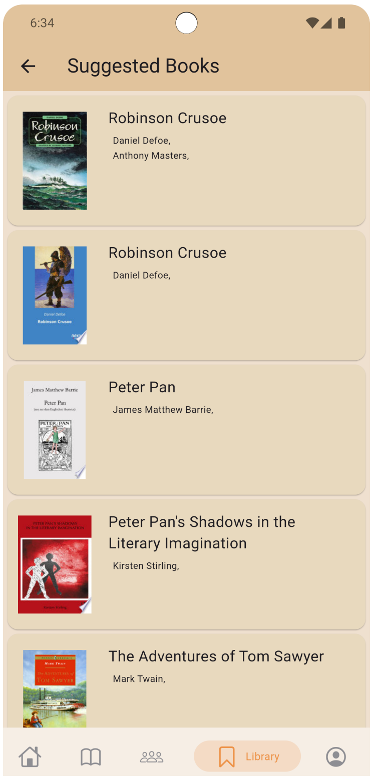
Harry Potter and the Chamber of Secrets
J. K. Rowling,

Treasure Island
Robert Louis Stevenson,

The black arrow
Robert Louis Stevenson,

Swiss Family Robinson
Johann David Wyss,

Figure 6.12: Favorite Books



Robinson Crusoe
Daniel Defoe,
Anthony Masters,

Robinson Crusoe
Daniel Defoe,

Peter Pan
James Matthew Barrie,

Peter Pan's Shadows in the Literary Imagination
Kirsten Stirling,

The Adventures of Tom Sawyer
Mark Twain,

Figure 6.13: Suggested Books

Displaying Suggested Books

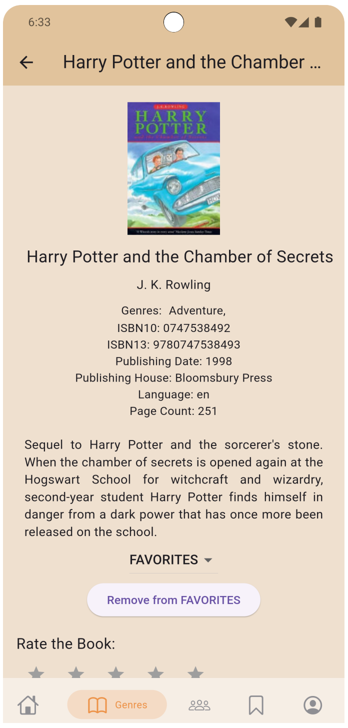
Once the AI-generated suggestions are processed, they are displayed on the Favorite Books screen as actual book entries rather than plain text. Each recommended book is presented with a visually appealing card containing the book cover, title and authors. This approach provides a more intuitive and engaging way for users to explore new books, making the recommendation system feel more integrated and dynamic.

This feature not only enhances book discovery but also ensures that users receive relevant, high-quality recommendations that evolve with their reading preferences. The provided screenshots, figures 6.12 and 6.13, illustrate the Favorite Books screen, the AI suggestion button, and the dynamically generated book recommendations.

6.4 Specific Book Screen

When the user clicks a specific book card, the individual page of that book is opened. On this page, where a picture of the book is displayed if available, all the book's details are shown, like Title, Author, Genres, ISBNs, Publishing Date and House, Language, and Page count. Below this information, there is also a summary of the book, which is retrieved from the database by the backend and displayed in the frontend.

Then, the application provides the option to put this book in one of three different lists: Favorites, Next, and Read. When the book is not in one of the lists, and that list is selected from the dropdown menu, the button reads Add to Favorites if the list is namely Favorites. Similarly, if the book is in the list, the button reads Remove from Favorites, and by clicking it, the book is removed from the Favorites list.



Harry Potter and the Chamber of Secrets

J. K. Rowling

Genres: Adventure,
ISBN10: 0747538492
ISBN13: 9780747538493
Publishing Date: 1998
Publishing House: Bloomsbury Press
Language: en
Page Count: 251

Sequel to Harry Potter and the sorcerer's stone. When the chamber of secrets is opened again at the Hogswart School for witchcraft and wizardry, second-year student Harry Potter finds himself in danger from a dark power that has once more been released on the school.

Figure 6.14: Book Screen 1/2

There is also the option to leave a review for the book. This first is provided by a rating option, where five stars are displayed. If the user wants to rate a book, the number of stars equal to the rating should be filled. After the stars are filled by clicking them, there is a text box where the user can freely write the review comment, and at the end, the button ”Submit Review” is available. If the user has already given a review for a specific book, then the option to leave the review is closed, and the button reads ”Review already submitted”, meaning that the user can not submit two reviews for a book.

An average rating of the book is shown in a decimal number format. This average rating is calculated based only on the ratings of the users in this application, so there is no connection to an external database for reviews. Then, at the very end of the page, all the reviews from the users are displayed, together with the username, the rating, and the date when the review was submitted.

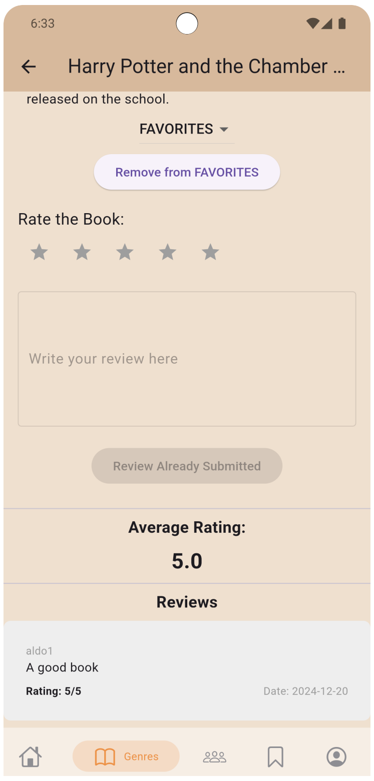


Figure 6.15: Book Screen 2/2

6.5 Profile Screen

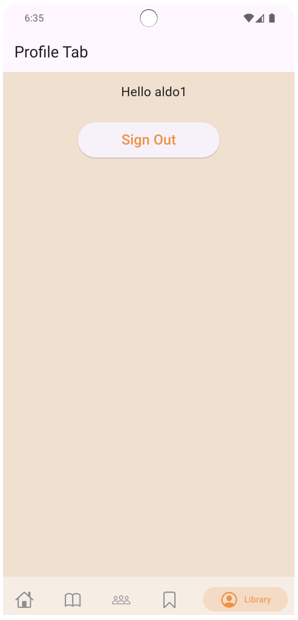


Figure 6.16: Profile Screen

The final page is the profile page, which contains the sign out option, so that the user can sign out and end the session. The user can then log in again, or log in with another account.

7 Test

7.1 Test Plan and Methodology

The advanced testing of software development is one of the most important stages because it is the stage that proves that the software is functional at the desired level, performs at the desired level, and gives a smooth user experience. The test process for this application involves implementing different testing techniques to confirm the system's varied functionality.

Testing Purposes

The primary purposes of the tests are:

- Ensuring functional correctness – To validate them, all the aspects of these features, e.g. user authentication, book recommendations, review submission, and book searching, need to be tested as expected.
- Validating API responses – The interaction with the backend, which includes working with OpenAI API and MySQL database through the API, will assure that correct and structured responses are sent to the client.
- Assessing UI responsiveness and usability – Confirm the system that is fast and responsive to user interactions by touching different devices and screen sizes through the Flutter frontend.
- Confirming data integrity and consistency – Overall, the ability of the database to receive, store, and retrieve data is crucial for the applications to run correctly and to be free from inconsistencies.
- Evaluating performance and scalability – Measure system response times and load-handling capabilities, especially for book recommendation queries.
- Identifying and resolving security vulnerabilities – Conduct tests to ensure user data is protected, authentication mechanisms are secure, and API endpoints prevent unauthorized access.

Test Methodology

The testing methodology uses both manual and automated testing techniques that are needed in order to get comprehensive coverage.

Implemented Tests:

- **Unit Testing:** The test concerning backend components such as the database queries, services, and controllers are unit tested with JUnit and Mockito. In other words, these unit tests help to make sure code functions correctly on its own.
- **API Testing:** The RESTful API services are tested with Postman to check that the requests and the responses are in the right format and that edge cases are handled appropriately.

Out of scope tests:

- **Integration Testing:** Integration tests check if both the Spring Boot backend and the MySQL database are working well with no errors. Spring Boot Test and TestContainers are used to simulate real-world interactions.
- **User Acceptance Testing (UAT):** Different operations are performed by system users, and experiences are recorded to gather usability, performance, and overall experience feedback.

7.2 Unit Testing

OpenAIService Test

Unit testing ensures the reliability of individual components in the application. The OpenAIServiceTest class focuses on testing the getBookSuggestions method, which interacts with OpenAI's API to generate book recommendations. To isolate the logic, Mockito is used to mock dependencies, including RestTemplate for making API requests and BookService for retrieving book details.

```java
@ExtendWith(MockitoExtension.class)
class OpenAIServiceTest {

    @Mock
    private RestTemplate restTemplate;

    @Mock
    private BookService bookService;

    @InjectMocks
    private OpenAIService openAIService;

    @BeforeEach
    void setUp() {
        ReflectionTestUtils.setField(openAIService, "bookService", bookService);
        ReflectionTestUtils.setField(openAIService, "model", "gpt-3.5-turbo");
        ReflectionTestUtils.setField(openAIService, "url", "https://api.openai.com/v1/
            chat/completions");
        ReflectionTestUtils.setField(openAIService, "key", "test-api-key");
    }
```

Listing 7.1: Mocked Components of OpenAIServiceTest

The test suite includes three main scenarios:

1. **Successful Response Handling** (Listing 7.2) :
   - Simulates a valid API response containing book suggestions.
   - Uses mocked data to retrieve book details from BookService.
   - Ensures the response contains the expected number of books with correct titles.
2. **Empty Response Handling** (Listing 7.3) :
   - Mocks an API response where no book suggestions are returned.
   - Verifies that the service correctly returns an HTTP 204 (No Content), ensuring the application can handle cases where no recommendations are available.
3. **Malformed Response Handling** (Listing 7.4) :
   - Tests the behavior when the API returns an unexpected response structure (e.g., missing expected fields).
   - Confirms that the method correctly identifies the malformed response and returns no results instead of failing.

```
 @Test
void testGetBookSuggestions_SuccessfulResponse() throws JsonProcessingException {
    // Mock favorite books
    List<String> favoriteBooks = List.of("Book A", "Book B");

    Map<String, Object> mockResponseBody = Map.of(
        "choices", List.of(
            Map.of(
                "message", Map.of(
                    "function_call", Map.of(
                        "arguments", "{\"titles\": [\"Book X\", \"Book Y\", \"Book
                            Z\", \"Book W\", \"Book V\"]}"
                    )
                )
            )
        )
    );

    ResponseEntity<Map> mockResponse = new ResponseEntity<>(mockResponseBody,
        HttpStatus.OK);
    doReturn(mockResponse).when(restTemplate).exchange(anyString(), eq(HttpMethod.
        POST), any(HttpEntity.class), eq(Map.class));

    // Mock book retrieval from BookService
    List<Book> mockBooks = List.of(
            new Book(1L, "Suggested Book 1", "img1.jpg", new HashSet<>(), new
                HashSet<>(), "1234567890", "9876543210123",
                    "2020-01-01", "Publisher A", "English", 300, "Summary 1"),
            new Book(2L, "Suggested Book 2", "img2.jpg", new HashSet<>(), new
                HashSet<>(), "1234567891", "9876543210124",
                    "2021-02-02", "Publisher B", "English", 350, "Summary 2")
    );
    when(bookService.getBooks(anyList())).thenReturn(mockBooks);

    // Call method
    ResponseEntity<List<BookDto>> result = openAIService.getBookSuggestions(
        favoriteBooks);

    // Verify results
    assertEquals(HttpStatus.OK, result.getStatusCode());
    assertEquals(2, result.getBody().size());
    assertEquals("Suggested Book 1", result.getBody().get(0).getTitle());
    assertEquals("Suggested Book 2", result.getBody().get(1).getTitle());
}
```

Listing 7.2: Successful Response Handling

```java
@Test
void testGetBookSuggestions_EmptyResponse() {
    Map<String, Object> responseBody = new HashMap<>();
    responseBody.put("choices", new ArrayList<>());

    ResponseEntity<Map> mockResponse = new ResponseEntity<>(responseBody,
        HttpStatus.OK);
    doReturn(mockResponse).when(restTemplate).exchange(anyString(), eq(HttpMethod.
        POST), any(HttpEntity.class), eq(Map.class));

    ResponseEntity<List<BookDto>> result = openAIService.getBookSuggestions(List.of
        ("Book A"));

    // Verify that no content is returned
    assertEquals(HttpStatus.NO_CONTENT, result.getStatusCode());
}
```

Listing 7.3: Empty Response Handling

```java
 @Test
void testGetBookSuggestions_MalformedResponse() {
    // Mock API returning a response with an invalid structure (missing "choices")
    Map<String, Object> malformedResponse = new HashMap<>();
    // The response body doesn't contain "choices", simulating a malformed response
    malformedResponse.put("unexpected_key", "unexpected_value");

    ResponseEntity<Map> mockResponse = new ResponseEntity<>(malformedResponse,
        HttpStatus.OK);
    doReturn(mockResponse).when(restTemplate).exchange(anyString(), eq(HttpMethod.
        POST), any(HttpEntity.class), eq(Map.class));

    ResponseEntity<List<BookDto>> result = openAIService.getBookSuggestions(List.of
        ("Book A"));

    // Verify that no content is returned due to the malformed response
    assertEquals(HttpStatus.NO_CONTENT, result.getStatusCode());
}
```

Listing 7.4: Malformed Response Handling

7.3 API Testing

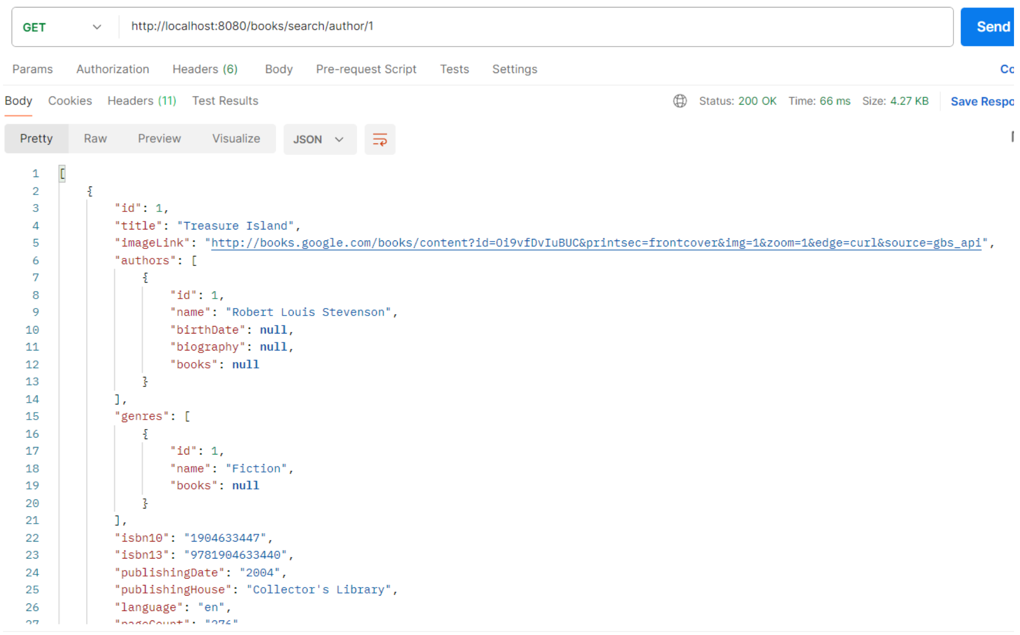
Backend stability and correctness are the most important parts of the application's development. In order to verify the functionality of the RESTful APIs, numerous tests were carried out using Postman, a program that is extensively used for API testing. Postman allows you to send requests, inspect responses, and make sure that the backend perfectly manages different scenarios, for instance, user authentication, book retrieval, and AI-driven recommendations. Furthermore, developers can prevent some issues, such as incorrect data format, improper request handling, and authentication failures, by performing single endpoint tests instead of combining the backend with the Flutter frontend before delivering the application to the client.

Book Retrieval APIs

To get the books from an author, the author id is used and a GET call is made to **http://localhost:8080/books/search/author/{id}** , where at the end the actual id of the author is inserted.

The response is constructed in such a way, that a list of BookDTOs is returned. This way the backend can send a whole book object to the frontend, thus allowing the frontend to show the books as cards and not just text.

The result from this API call looks exactly like this:

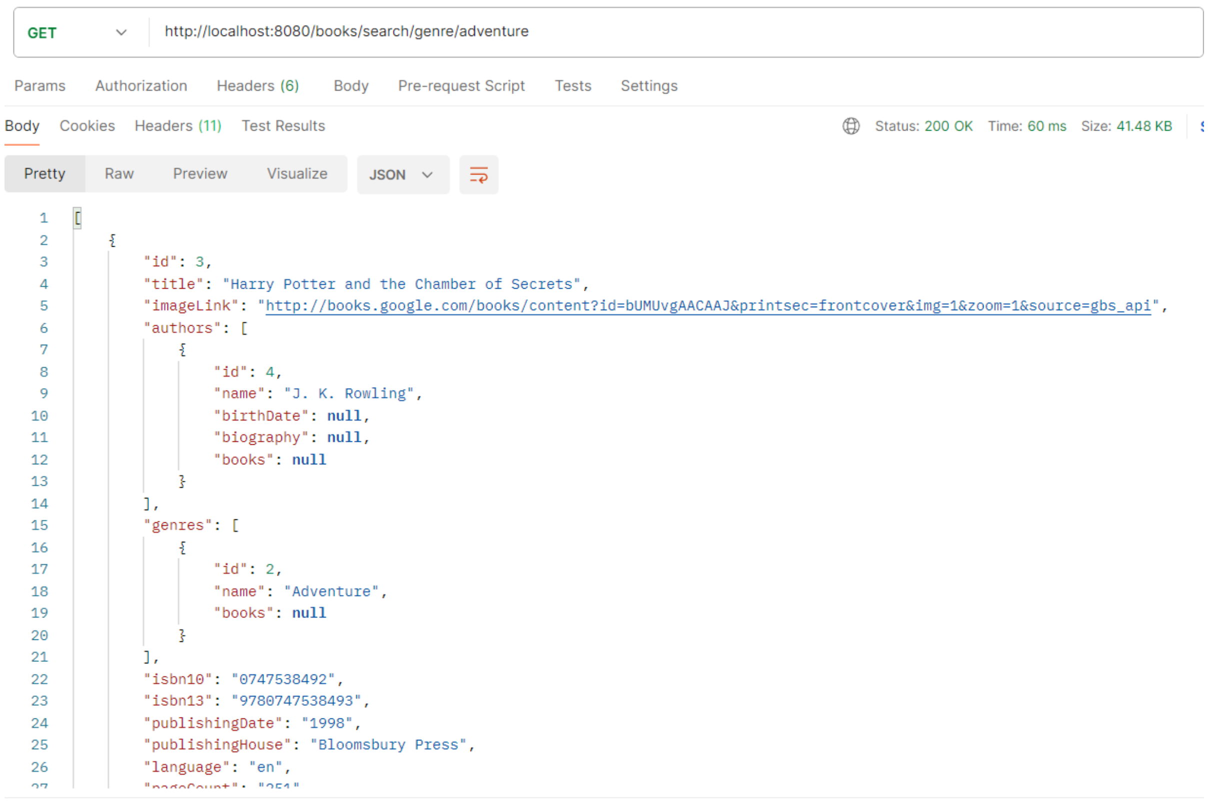


```
GET http://localhost:8080/books/search/author/1
Status: 200 OK   Time: 66 ms   Size: 4.27 KB
[
    {
        "id": 1,
        "title": "Treasure Island",
        "imageLink": "http://books.google.com/books/content?id=0i9vfDvIuBUC&printsec=frontcover&img=1&zoom=1&edge=curl&source=gbs_api",
        "authors": [
            {
                "id": 1,
                "name": "Robert Louis Stevenson",
                "birthDate": null,
                "biography": null,
                "books": null
            }
        ],
        "genres": [
            {
                "id": 1,
                "name": "Fiction",
                "books": null
            }
        ],
        "isbn10": "1904633447",
        "isbn13": "9781904633440",
        "publishingDate": "2004",
        "publishingHouse": "Collector's Library",
        "language": "en",
```

Figure 7.1: The GET Call to retrieve books for the author with id = 1

Figure 7.1 shows clearly how the response is formatted, even though this is just a part of the response containing only the first book. The other books that belong to this author are in continuation of the response, and they are formatted the same.

To retrieve the books from a specific genre, the logic is the same as for the author, but the difference is that in this case the name of the genre is being used instead of its id, namely to get the adventure books the call is **http://localhost:8080/books/search/genre/adventure**. The response is exactly the same as in the authors' case.

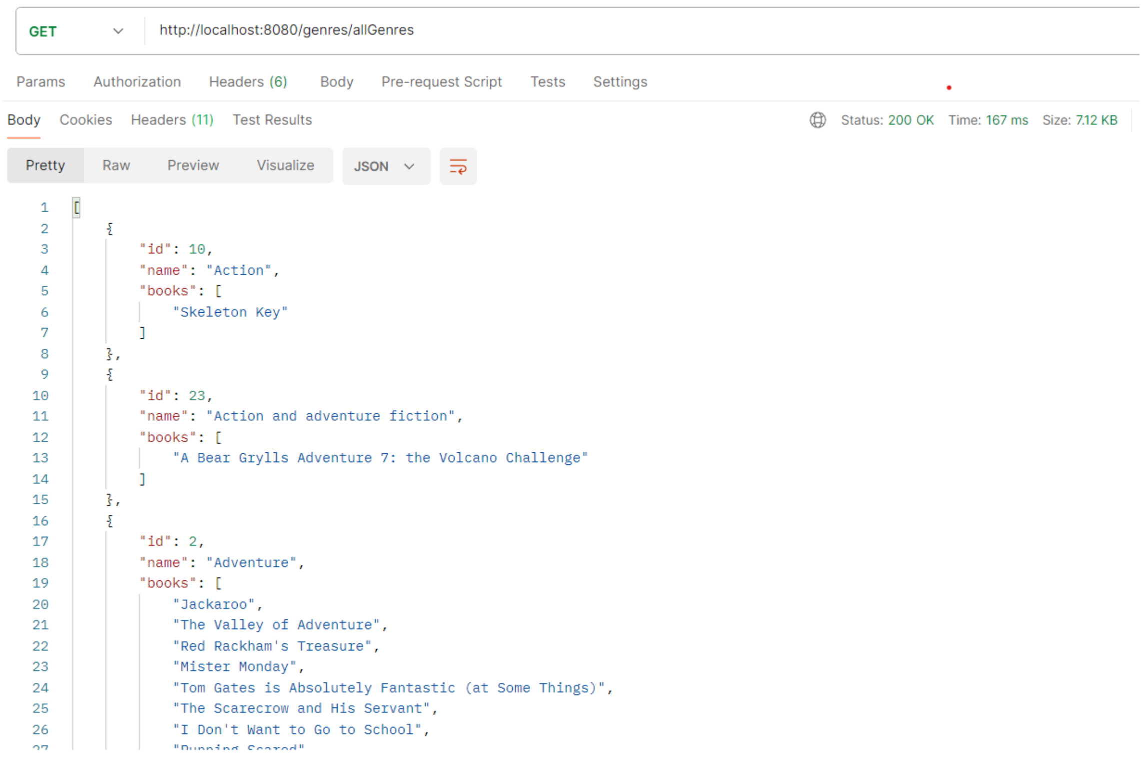


```
[
    {
        "id": 3,
        "title": "Harry Potter and the Chamber of Secrets",
        "imageLink": "http://books.google.com/books/content?id=bUMUvgAACAAJ&printsec=frontcover&img=1&zoom=1&source=gbs_api",
        "authors": [
            {
                "id": 4,
                "name": "J. K. Rowling",
                "birthDate": null,
                "biography": null,
                "books": null
            }
        ],
        "genres": [
            {
                "id": 2,
                "name": "Adventure",
                "books": null
            }
        ],
        "isbn10": "0747538492",
        "isbn13": "9780747538493",
        "publishingDate": "1998",
        "publishingHouse": "Bloomsbury Press",
        "language": "en",
```

Figure 7.2: Results for the GET Call to retrieve books for the Adventure genre

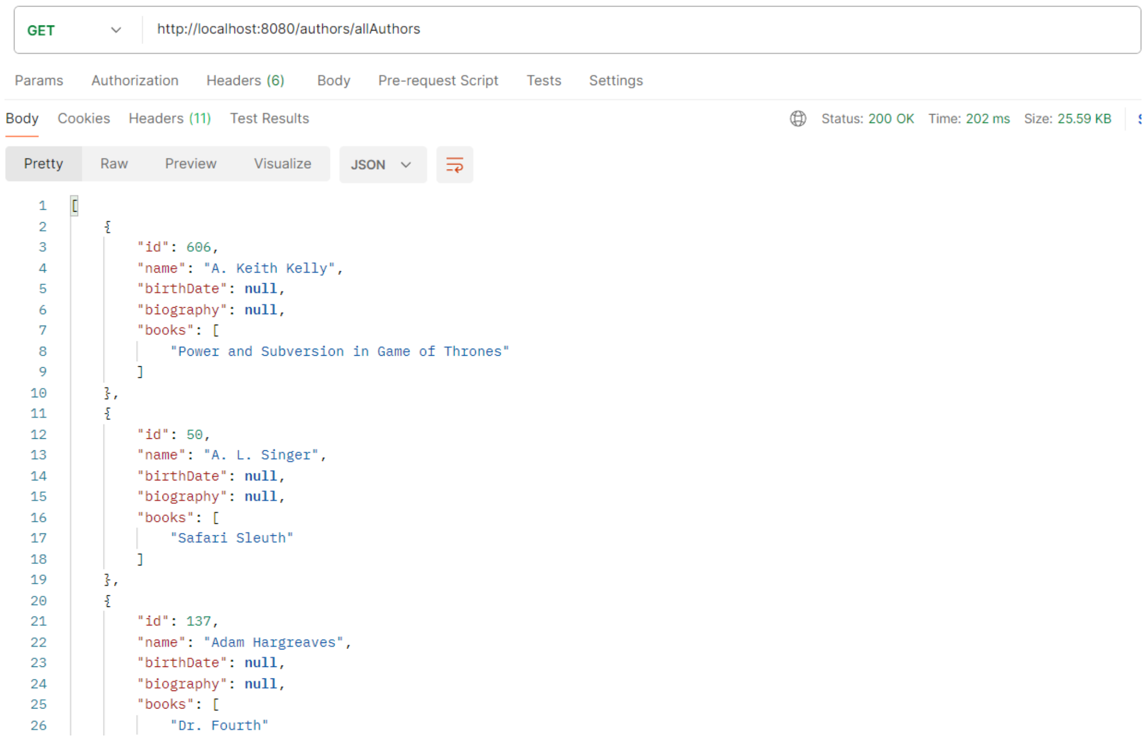
Authors and Genres Retrieval APIs

To allow the frontend to show all the authors and all the genres in their respective pages, the backend allows for GET Calls to **http://localhost:8080/authors/allAuthors** and **http://localhost:8080/genres/allGenres**. The result in both cases is a list of all authors or all genres in alphabetical order containing some information about the author or the genre, and the books that belong to it. This is a key feature because this way the frontend can show all the books that belong to an author or genre when the user clicks on one.



```
[
    {
        "id": 10,
        "name": "Action",
        "books": [
            "Skeleton Key"
        ]
    },
    {
        "id": 23,
        "name": "Action and adventure fiction",
        "books": [
            "A Bear Grylls Adventure 7: the Volcano Challenge"
        ]
    },
    {
        "id": 2,
        "name": "Adventure",
        "books": [
            "Jackaroo",
            "The Valley of Adventure",
            "Red Rackham's Treasure",
            "Mister Monday",
            "Tom Gates is Absolutely Fantastic (at Some Things)",
            "The Scarecrow and His Servant",
            "I Don't Want to Go to School",
```

Figure 7.3: Results for the GET Call to retrieve all the genres



```
[
    {
        "id": 606,
        "name": "A. Keith Kelly",
        "birthDate": null,
        "biography": null,
        "books": [
            "Power and Subversion in Game of Thrones"
        ]
    },
    {
        "id": 50,
        "name": "A. L. Singer",
        "birthDate": null,
        "biography": null,
        "books": [
            "Safari Sleuth"
        ]
    },
    {
        "id": 137,
        "name": "Adam Hargreaves",
        "birthDate": null,
        "biography": null,
        "books": [
            "Dr. Fourth"
```

Figure 7.4: Results for the GET Call to retrieve all the authors

Lists Retrieval APIs

To retrieve the books, authors or genres that a user has put on a list, GET calls must be made from the frontend to the backend. In this section, only the retrieval of the FAVORITE books list is shown, as the logic stands the same for NEXT and READ book lists, as well as favorite authors and genre list.

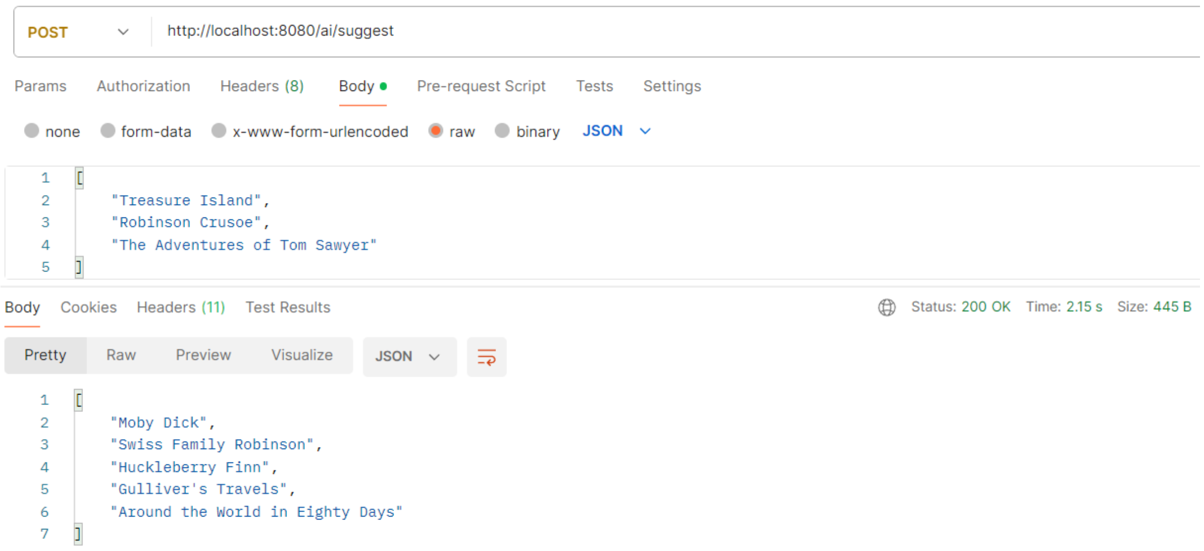
To get the favorite books of a specific user, in this example user with id 1, the URL of the call is: **http://localhost:8080/user/1/bookList/FAVORITES/books**. This URL shows that the call must be made with the user id and the name of the list on it. In the two other cases, instead of FAVORITES there should be NEXT or READ. The result of this call is again a JSON with a list of the bookDTOs, to allow once more the frontend to display book cards.

This GET Call is really important because it facilitates the way the AI suggestions are created. The frontend send exactly the titles of these books to the backend to allow it to get the AI suggestions by making this time a POST Call to the OpenAI API.

AI Suggestions API Test

To test the AI functionality of the app, POST calls are made to specifically this URL: **http://localhost:8080/ai/suggest**. Unlike the other cases where the arguments were included in the URL, this time the arguments are given to the Body via the RequestBody annotation of SpringBoot: **@RequestBody List<String> favoriteBooks**. To provide these books to the POST call, there is the JSON format body option in Postman. In both cases that are to be displayed, a list of three books is being used: ”Treasure Island”, ”Robinson Crusoe”, ”The Adventures of Tom Sawyer”.

The first thing to be tested was the plain AI response to the input given. When instructed to suggest books based on the input, the AI returns a JSON containing only the titles of the books. For the above list of books, one suggestion of the OpenAI API was: ”Moby Dick”, ”Swiss Family Robinson”, ”Huckleberry Finn”, ”Gulliver’s Travels”, ”Around the World in Eighty Days”. Of course, each time it is called, it gives different results, even though the same books are given as input, and that is one of the main features.


POST http://localhost:8080/ai/suggest

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
[
    "Treasure Island",
    "Robinson Crusoe",
    "The Adventures of Tom Sawyer"
]
```

Body Cookies Headers (11) Test Results   Status: 200 OK Time: 2.15 s Size: 445 B

Pretty Raw Preview Visualize JSON

```
[
    "Moby Dick",
    "Swiss Family Robinson",
    "Huckleberry Finn",
    "Gulliver's Travels",
    "Around the World in Eighty Days"
]
```

Figure 7.5: Results for the POST Call to the AI by providing a JSON body

This list of books provided by the AI is taken, manipulated, and in the end it is shown as book cards in the frontend. This is all explained in the Implementation chapter and shown as example in the Demonstration chapter. The result is a JSON with the list of the books as bookDTOs.

This way the backend communicates with the OpenAI API and sends the wanted response to the call that the frontend actually makes to retrieve the suggested books from the backend.

8 Summary and Future Improvements

8.1 Key Takeaways and Accomplishments

The successful development of this application has clearly shown how modern technologies can be used in the making of an engaging and easy-to-use book recommendation system. Using Flutter as the frontend, Spring Boot as the backend, MySQL as the database, and OpenAI as the software that provides the AI-driven suggestions, this initiative shows a scalable and modular architecture that allows for the smooth running of the system with ease of maintenance.

- The successful implementation of a personalized book recommendation system is the most important achievement of this project. Through AI-driven recommendations inclusion, the users can be presented with the book suggestions that fit their liking, hence the experience of reading is improved. The system's use of OpenAI's "function calling" ensures that book recommendations are structured, accurate, and seamlessly processed by the backend. The system in this way avoids inconsistencies, and it only gives meaningful suggestions that are helpful to the users.
- Yet another milestone is the strong authentication system, which comprises user registration, email verification, and secure login mechanisms. This ensures that only authorized users can access the system, enhancing security and user trust.
- Technically, the project implements a layered architecture by ensuring a clear separation of concerns between the Controller, Service, and Repository layers. This type of architecture means that the program's code is easier to update, run reliably, and is scalable. At the same time, it is very easy to implement new features. In addition, API testing through Postman has resulted in the backend working as it is supposed to, meaning it can produce responses correctly and integrate smoothly with the frontend.
- Moreover, the effective utilization of Flutter's UI has given a great and quick interface that has produced a user experience of cross-platform devices and screen sizes with no disruptions.

On the whole, state-of-the-art technologies have been used in this project to create an efficient, scalable, and user-centric book recommendation system. The breakthroughs in backend development, fronted implementation, AI integration, and database design are the basis of further improvements, which specify the possibilities of AI-driven applications in the delivery of personalized content.

8.2 Future Improvements and Scope For Expansion

In addition, the current version of the book suggestion application has a very solid structure for the fact that personalized book discovery is made possible, but there are several areas for future enhancements and expansion, with the objective of providing a seamless user experience, data accuracy improvement, and overall application capabilities expansion to allow a more complete and pleasurable platform for bookaholics to use.

1. One significant improvement would be the introduction of author-based recommendations. It has been up to date that book suggestions come up based on users' favorite books, but a new dimension of personalization could be included that could be done by incorporating the favorite authors. This utility would be the place where not only will the users get recommendations like those of their favorite books, but they can also be suggested by authors whose genres and themes match their tastes.

2. Another potential enhancement is perfecting the authentication system. Even though the current one secures the safety of users and passes the verification procedure, there are other features that can be implemented, like user data change (email, password, username). Also by incorporating third-party authentication options such as Google Sign-In, the process will not only be simplified, but also the interface will be seamless.

3. Moreover, the database can be added with a more extensive book inventory, a necessary step that enables not only the accuracy but also the variety of the recommendations. Having a huge amount of detailed book metadata, such as new and old ones, would be a way to make the library broader and much more different in terms of its suggestions.

4. The application can become a social reading platform in the future where the readers can interact with each other by following one another, sharing their reviews, and talking about their favorite books. Introducing interactive features such as user-generated book lists, discussion forums, and real-time reading progress tracking would change the application into a vibrant book lovers' community.

Overall, these future enhancements would greatly expand the scope and usability of the application, making it not just a book recommendation tool, but a comprehensive literary hub for readers worldwide.

8.3 Conclusion and Closing Remarks

In conclusion, the development of this book recommendation application has been a rewarding and enriching experience. This project has really provided me with an opportunity to get involved at the crossroads of personalizing user experiences and artificial intelligence, through which I was able to build an application that gives dynamic book recommendations. At every step of the process, I have come to learn how complex systems can be designed, developed, and fine-tuned to meet user needs and bring real value.

The development journey has widened my technological horizons far beyond software knowledge, and my problem-solving skills have also widened. I was taught valuable lessons about the importance of perseverance, creativity, and the ability to iterate on ideas through implementing AI features, seamless backend-frontend communication, and the refining of recommendation algorithms. The experience was great evidence to put theory into practice.

On the whole, the results of this project will be a good benefit for future endeavors. This project has not only provided me with practical experience in problem-solving and software development but also has kindled an interest in investigating AI-driven applications and customized systems even more. In the future, I will convey the knowledge I have learned and apply it to new problems and tasks. I will make sure that I develop both my technical and professional skills as the experience develops.

Bibliography

[1] Analytics Vidhya: *All You Need to Know about Recommendation Systems*. https://www.analyticsvidhya.com/blog/2022/01/all-you-need-to-know-about-recommendation-systems/. 2022

[2] Astera: *REST-API-Definition: Was sind REST-APIs (RESTful-APIs)?* https://www.astera.com/de/type/blog/rest-api-definition/. 2024

[3] Campesato, Oswald: *Java for Developers Pocket Primer*. Stylus Publishing, LLC, 2022

[4] GeeksForGeeks: *What is Flutter?* https://www.geeksforgeeks.org/what-is-flutter/. 2024

[5] Javin Paul: *Top 15 Spring Data JPA Interview Questions with Answers*. https://www.java67.com/2021/01/spring-data-jpa-interview-questions-answers-java.html. 2021

[6] Napoli, Marco L.: *Beginning flutter: a hands on guide to app development*. John Wiley & Sons, 2019

[7] OpenAI: *OpenAI API: Overview*. https://platform.openai.com/docs/overview. 2025

[8] OpenAI: *The power of personalized AI*. https://openai.com/global-affairs/the-power-of-personalized-ai/. 2025

[9] Richards, Mark: *Fundamentals of Software Architecture*. O'REILLY, 2020

[10] Romain GARCIA: *JWT tokens and security – working principles and use cases*. https://www.vaadata.com/blog/jwt-tokens-and-security-working-principles-and-use-cases/. 2016

[11] Roy Thomas Fielding: *Architectural Styles and the Design of Network-based Software Architectures*. https://ics.uci.edu/~fielding/pubs/dissertation/top.htm. 2000

[12] Suehring, Steve: *MySQL Bible*. Wiley Publishing, Inc., 2002

[13] Tarun Sharma: *OpenAI - Function Calling*. https://www.linkedin.com/pulse/azure-openai-function-calling-tarun-sharma/. 2023

[14] Tegetmeier, C., Johannssen, A. Chukhrova, N.: *Artificial Intelligence Algorithms for Collaborative Book Recommender Systems. Ann. Data. Sci. 11, 1705–1739*. https://doi.org/10.1007/s40745-023-00474-4. 2024

[15] Walls, Craig: *Spring Boot in action*. Manning, 2016

[16] Zhang, Q., Lu, J. Jin, Y.: *Artificial intelligence in recommender systems. Complex Intell. Syst. 7, 439–457*. https://doi.org/10.1007/s40747-020-00212-w. 2021

[17] Zohud, Tasnim ; Zein, Samer: Cross-platform mobile app development in industry: A multiple case-study. In: *International Journal of Computing* 20 (2021), Nr. 1, S. 46–54

A Abbreviations

API Application Programming Interface

CRUD Create, Read, Update, Delete

JWT JSON web token

SDK Software Development Kit

AI Artificial Intelligence

HTTP Hypertext Transfer Protocol

iOS iPhone operating system

JPA Java Persistence API

UI User Interface

JSON JavaScript Object Notation

REST Representational State Transfer

DBMS Database Management System

RDBMS Relational Database Management System

ACID Atomicity, Consistency, Isolation, and Durability

LTS Long Term Support

IDE Integrated Development Environment

DTO Data Transfer Object

SQL Structured Query Language

B Acknowledgements

I would like to express my deepest gratitude to my thesis advisors, **Prof. Dr. Martin Lapke** and **Prof. Dr. Marc Hensel**, for their invaluable guidance, unwavering support, and insightful feedback throughout this journey. Their expertise and encouragement have been instrumental in shaping both my research and my growth as a student.

I am also grateful to my university and department for providing the resources and academic environment necessary for this work. A special thank you to fellow students, whose discussions and advice have made this process both enriching and enjoyable.

Beyond academia, I owe immense appreciation to my friends and family for their constant encouragement, patience, and belief in me, even during the most challenging moments. Their support has been a driving force in completing this thesis.

Finally, I acknowledge the dedication and effort that went into this project, and I am proud to have reached this milestone. This work represents not only an academic achievement but also a personal journey of perseverance and learning.

Declaration

I declare that this Bachelor Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used.

City	Date	Signature